CSE 341 Sample Midterm #4

1. Expressions

For each ML expression in the left-hand column of the table below, indicate in the right-hand column its value. Be sure to (e.g., `7.0` rather than `7` for a `real`; `String`s in quotes e.g. `"hello"`; `true` or `false` for a `bool`). Assume that the following value has been declared:

```
val numbers = [7.3, 1.4, 3.2, 4.9, 2.2];
```

Expression	Value
a) `reduce(op +, numbers)`	
b) `mapx(hd o tl o rev, [1--3, 2--4, 5--6, 3--9])`	
c) `reduce(op *, mapx(round, numbers))`	
d) `filterx(fn x => 20 div x < 2, (1--12) @ (8--14))`	
e) `mapx(fn x => x--4, 2--5)`	
f) `reduce(op *, mapx(fn x => 3 * x - 1, 1--3))`	
g) `mapx(fn n => n - 1.1, numbers)`	
h) `reduce(op +, filterx(fn n => 6 mod n = 0, 1--5))`	
i) `mapx(fn x => (x, x - 1), 2--4)`	
j) `implode(reduce(op @, mapx(tl o explode, ["foo","bar","baz"])));`	

2. Types

For each ML expression in the left-hand column of the table below, indicate its **type** in the right-hand column assuming the expression was added to the top-level environment. Assume that the following value has been declared:

```
val lst = [2.8, 14.7];
```

Expression	Type
a) `lst`	
b) `(lst, mapx(round, lst))`	
c) `mapx(fn x => (round(x), x), lst)`	
d) `round o hd o tl`	
e) `fn x => [(x, [x])]`	

3. Scope

What value does the variable `answer` store after executing the following code?

```
val n = 2;
val x = 4;
val y = 6;
val z = 8;
fun f(n) =
    let val x = n + 10
        val y =
            let val n = x + 20
                val y = 15
            in  n + y
            end
        val z = n + z
    in  x + y + z + n
    end;
val answer = f(n + 1);
```

CSE 341 Sample Midterm #4

4. Curried Functions

For this problem, in addition to being able to call the functions listed on the front page of the test, you may call the function `curry` and the curried versions of `map`, `filter` and `List.foldl/r`) that we used in Homework #3, as well as the following curried functions:

- `fun curry f x y`
  Returns a curried version of the 2-argument function/operator `f`
- `fun map2 f list`
  curried version of `map` written in lecture
- `fun filter2 f list`
  curried version of `filter` written in lecture
- `fun reduce2 f list`
  curried version of `reduce` written in lecture

As in homework #3, you may use only `val` definitions to solve the following problems. You may not use `fun` or `fn` definitions to define a function.

a) Define a function `sum` that takes a nonempty string as an argument and that returns the sum of the ordinal values of the characters in the string. For example, `sum("abc")` should return 294, which is the sum of the three ordinal values of the characters (97 + 98 + 99). Recall that the function `ord` returns the ordinal value of a character.

b) Define a function `counts` that takes a nonnegative integer n as an argument and that returns a list of n lists containing the consecutive integers 1 through 1, 1 through 2, 1 through 3, and so on, ending with 1 through n. For example:

- `counts(0)` should return `[]`
- `counts(3)` should return `[[1],[1,2],[1,2,3]]`
- `counts(5)` should return `[[1],[1,2],[1,2,3],[1,2,3,4],[1,2,3,4,5]]`

CSE 341 Sample Midterm #4

5. Functions

Write a function `evens` that takes a list of integers as an argument and that returns the number of even numbers in the list. For example:

- `evens(1--9)` should return `4`
- `evens([3, 19, 4, 2, 17, 8, 1, 6, 9, 42])` should return `5`

You are not allowed to call higher order functions to solve this problem (`map`, `filter`, `reduce`, etc). Your function should run in O(n) time where n is the length of the list and should be **tail-recursive**.

6. Functions

Write a function `mapIndex` that takes a function and a list as arguments and that returns the result of applying the function to each value in the list along with its index. This function is a slight variation of map. The function passed to `mapIndex` takes a tuple containing a value from the list and the index of that value. We will use zero-based indexing where the first value has index 0, the second value has index 1, and so on. For example, given a list of ints, you can multiply each value in a list by its index with a call like this:

- `mapIndex(fn (x, i) => x * i, 1--5)` should return `[0,2,6,12,20]`

Given a list of strings, you can add the index of each value to its size with a call like this:

- `mapIndex(fn (x, i) => size(x) + i, ["foo", "bar", "baz"])` should return `[3,4,5]`

Your function should run in O(n) time where n is the length of the list.

CSE 341 Sample Midterm #4

7. Functions

Write a function insert that takes a value and a list of lists as arguments and that returns the list obtained by inserting the value at the front of each list within the list of lists. For example:

- `insert(3, [1--3, 2--5, 5--7])` should return `[[3,1,2,3],[3,2,3,4,5],[3,5,6,7]]`
- `insert("foo", [["to", "be"], ["four", "score", "and"], []])` should return `[["foo","to","be"],["foo","four","score","and"],["foo"]]`

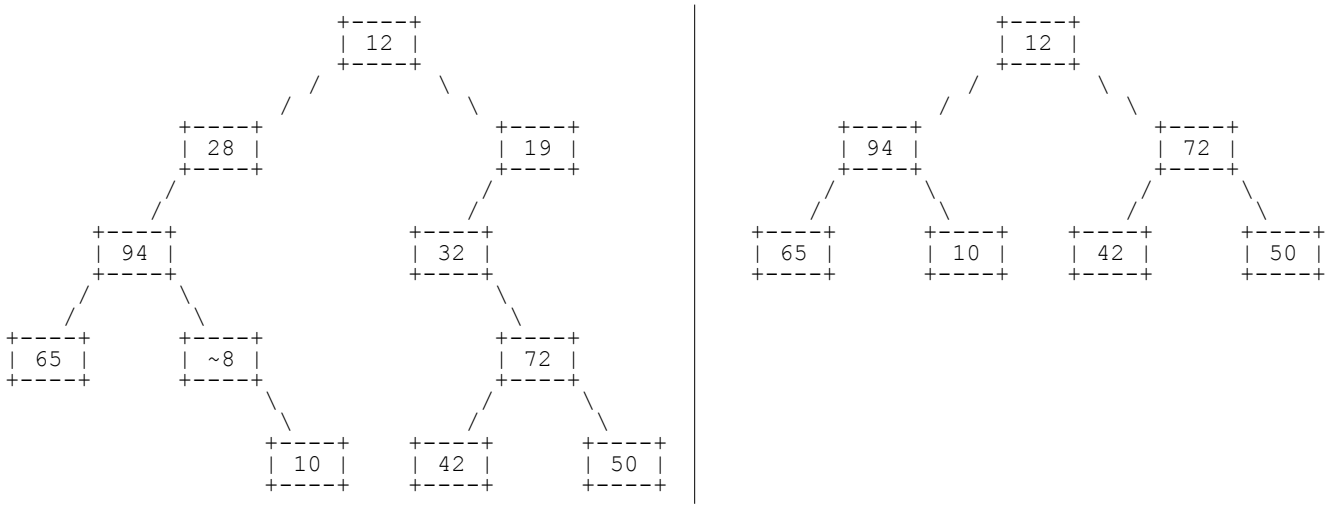
Your function should run in O(n) time where n is the length of the list.

8. Datatypes

Recall the `IntTree` data type we discussed in lecture for storing a binary tree of integers:

```
datatype IntTree = Empty | Node of int * IntTree * IntTree;
```

Write a method tighten that returns the tree obtained by eliminating branch nodes that have only one child from a binary tree of integers. For example, if a variable called `t` stores the tree below at left, then the call `tighten(t)` should return the tree on the right:



Notice that the nodes that stored the values 28, 19, 32, and ~8 have all been eliminated from the tree because each had one child. When a node is removed, it is replaced by its child. Notice that this can lead to multiple replacements because the child might itself be replaced (as in the case of 19 which is replaced by its child 32 which is replaced by its child 72).

CSE 341 Sample Midterm #4

9. Functions

a) Write a function `indexSum` that takes a list of `int lists` as an argument and that returns an index-weighted sum of the number of even values in each sublist. In particular, if the list is:

- `[lst0, lst1, lst2, ..., lst-n]`

then the function should return:

- `0 * (# evens in lst0) + 1 * (# evens in lst1) + 2 * (# evens in lst2) + 3 * (# evens in lst3) + ... + n * (# evens in lst-n)`

For example, given this value:

```
val test = [[1, 2, 3],[2, 4, 5, 6],[1, 0, 14, 12, 8, 9],[1, 3],[2, 4]];
```

the call `indexSum(test)` should return 19 computed as:

```
0 * 1 + 1 * 3 + 2 * 4 + 3 * 0 + 4 * 2
```

The function should return 0 if passed an empty list.

b) Write a function `minSum` that given a list of int lists will return a list with the sublists ordered so as to minimize the value of `indexSum` for the list. For example, given the variable test defined above:

- `minSum(test)` should return `[[1,0,14,12,8,9],[2,4,5,6],[2,4],[1,2,3],[1,3]]`

In this case, the result has an `indexSum` of 10. The function can change the order of the sublists in the result, but not the contents of the sublists. There might be more than one correct answer in which case your function can return any of the correct answers.

10. Functions

Write a function `pick` that takes an integer n and a list of values as arguments and that returns a list of all ways to pick n values from the list. For example:

- `pick(2, 1--5)` could return
  `[[4,5],[3,5],[3,4],[2,5],[2,4],[2,3],[1,5],[1,4],[1,3],[1,2]]`
- `pick(3, ["a", "b", "c", "d"])` could return
  `[["a","b","c"],["a","b","d"],["a","c","d"],["b","c","d"]]`

You may assume that the list has no duplicates. You may list the solutions in any order. There are no ways to pick n values from a list of length less than n. There is exactly one way to pick 0 elements from any list. As the number of values to pick gets large, this function may end up taking up a lot of time to execute, but it should be reasonably efficient when the number of values to pick is small. You may assume that the integer passed to the function is not negative.

CSE 341 Sample Midterm #4
Solutions

1. Expressions

Expression	Value
a) `reduce(op +, numbers)`	`19.0`
b) `mapx(hd o tl o rev, [1--3, 2--4, 5--6, 3--9])`	`[2,3,5,8]`
c) `reduce(op *, mapx(round, numbers))`	`210`
d) `filterx(fn x => 20 div x < 2, (1--12) @ (8--14))`	`[11,12,11,12,13,14]`
e) `mapx(fn x => x--4, 2--5)`	`[[2,3,4],[3,4],[4],[]]`
f) `reduce(op *, mapx(fn x => 3 * x - 1, 1--3))`	`80`
g) `mapx(fn n => n - 1.1, numbers)`	`[6.2,0.3,2.1,3.8,1.1]`
h) `reduce(op +, filterx(fn n => 6 mod n = 0, 1--5))`	`6`
i) `mapx(fn x => (x, x - 1), 2--4)`	`[(2,1),(3,2)(4,3)]`
j) `implode(reduce(op @, mapx(tl o explode, ["foo","bar","baz"])));`	`"ooaraz"`

2. Types

Expression	Type
a) `lst`	`real list`
b) `(lst, mapx(round, lst))`	`real list * int list`
c) `mapx(fn x => (round(x), x), lst)`	`(int * real) list`
d) `round o hd o tl`	`real list -> int`
e) `fn x => [(x, [x])]`	`'a -> ('a * 'a list) list`

3. Scope

```
val answer = 75
```

4. Curried Functions

```
val sum = reduce2 op+ o map ord o explode;
val counts = map (curry op-- 1) o curry op-- 1;
```

5. Functions

```
fun evens(lst) =
    let fun helper([], count) = count
        |   helper(x::xs, count) =
                if x mod 2 = 0 then helper(xs, count + 1)
                else helper(xs, count)
    in helper(lst, 0)
    end;
```

6. Functions

```
fun mapIndex(f, lst) =
    let fun helper([], n) = []
        |   helper(x::xs, n) = f(x, n) :: helper(xs, n + 1)
    in helper(lst, 0)
    end;
```

CSE 341 Sample Midterm #4
Solutions (continued)

7. Functions

```
fun insert(x, []) = []
|   insert(x, y::ys) = (x::y)::insert(x, ys);
```

8. Data Types

```
fun tighten(Empty) = Empty
|   tighten(Node(root, Empty, Empty)) = Node(root, Empty, Empty)
|   tighten(Node(root, left, Empty)) = tighten(left)
|   tighten(Node(root, Empty, right)) = tighten(right)
|   tighten(Node(root, left, right)) =
        Node(root, tighten(left), tighten(right));
```

9. Functions

a) Three possible solutions are shown.

```
fun indexSum(lst) =
    let fun helper([], n) = 0
        |   helper(x::xs, n) = n * evens(x) + helper(xs, n + 1)
    in helper(lst, 0)
    end;

fun indexSum([]) = 0
|   indexSum(lst) =
    reduce(op+, mapIndex(fn (x, n) => evens(x) * n, lst));

fun indexSum(lst) =
    foldl op+ 0 (mapIndex(fn (x, n) => evens(x) * n, lst));
```

b)

```
fun minSum(lst) = quicksort(fn (x, y) => evens(x) > evens(y), lst);
```

10. Functions

```
fun pick(0, lst) = [[]]
|   pick(n, []) = []
|   pick(n, x::xs) = insert(x, pick(n - 1, xs)) @ pick(n, xs);
```