CSE 341, Autumn 2010
Midterm Exam, Wednesday, November 3, 2010

Name: _____

Section: _____ **TA:** _____

Student ID #: _____

Score summary: (for grader only)

Problem	Description	Earned	Max
1	Expressions		14
2	Types		10
3	Scope / Closures		6
4	Curried Functions		15
5	Functions		10
6	Functions		10
7	Functions		10
8	Data Types		15
9	Functions		10
TOTAL	**Total Points**		**100**

Exam Rules

General Exam Guidelines:

- You have **50 minutes** to complete this exam. You must stop working once the instructor calls for papers. You may receive a deduction if you keep working after the instructor calls for papers.
- The exam is open-book/notes. You must work alone and may not use any computing devices including calculators. Cell phones, music players, and other electronics may NOT be out during the exam for any reason.
- Please be quiet during the exam. If you have a question or need, please raise your hand.
- Corrections or clarifications to the exam will be written at the front of the room.
- Please obey the University Code of Conduct during the exam.
- When you have finished the exam, please turn in your exam quietly and leave the room.

Programming Guidelines:

Unless otherwise noted, you may call any of the functions available in the standard top-level environment including:

- the standard operators (`~`, `+`, `-`, `*`, `/`, `div`, `mod`, `::`, `@`, `^`, `o`, `not`, `andalso`, `orelse`, `<`, `>`, `<=`, `>=`, `=`, `<>`)
- the numeric function `abs`
- the list functions `hd`, `tl`, `length`, `rev`, `foldl`, `foldr`
- the conversion functions `real`, `trunc`, `floor`, `ceil`, `ord`, `chr`, `str`
- the string functions `implode`, `explode`, `concat`, `size`
- the standard tuple functions `#1`, `#2`, etc.
- any functions from ML basic data type structures such as `Int`, `Real`, `String`, `Bool`, and `Char` (but not `List`)

You also may call any function that is included as a problem on this exam, whether or not you correctly solve that problem. You may call the functions described below that that we have discussed:

- `fun quicksort(f, list)`
  Returns the result of sorting the given list using the given comparison function where `f(a, b)` indicates that `a < b` (runs in O(n log n) time)
- `fun m -- n`
  Returns a list of the sequential integers from m to n inclusive; returns an empty list if `m > n`
- `fun mapx(f, list)`
  Returns the list obtained by applying f to every value of the given list
- `fun filterx(f, list)`
  Returns a list of the values from the given list for which f(a) is true
- `fun reduce(f, list)`
  For a list [x1, x2, x3, ..., xn], returns x1 for a list of length 1, otherwise returns f(x1, f(x2, f(x3, ..., f(xn-1, xn)...)))

You don't need to write any `use` or `open` statements in your exam code. Do not abbreviate any code on the exam.

Unless otherwise specified, you can write your own helper functions, and you can use functions of the basic type structures such as `Int`, `Real`, `String`, etc., but you are not allowed to use library functions like `List.nth` or `List.last` unless they are defined in the top-level environment. Helper functions must be defined with a `let` so that they do not become part of the top-level environment.

Unless this page or the problem mentions otherwise, your code you write will be graded purely on external correctness (proper behavior and output) and not on internal correctness (style). So, for example, redundancy or lack of comments will not reduce your score. Some functions do have specific style constraints, such as demanding a curried function.

Good luck!

1. Expressions

For each ML expression in the left-hand column of the table below, indicate its **value** in the right-hand column. 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 lst = [(1, 5), (~4, ~4), (2, 8), (~1, 2), (~3, ~5)];
```

Expression	Value
`a) hd(tl(lst))`	_____
`b) reduce(op *, 1--4)`	_____
`c) map (fn x => #2(x)) lst`	_____
`d) List.filter (fn(a, b) => a >= b) lst`	_____
`e) map (abs o Int.max) lst`	_____
`f) map (curry op-- 2) (3--5)`	_____
`g) reduce(op @, map (fn(a, b) => [a, b]) lst)`	_____

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 x = ("Suzy", "Smith", 27);
```

Expression	Type
`a) x`	_____
`b) (2.0, x)`	_____
`c) fn(a, b) => explode(hd(b) ^ a)`	_____
`d) fn(x, y) => [x, y]`	_____
`e) fn(x, y) => ((y, ~y), trunc(x))`	_____

3. Scope / Closures

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

```
val a = 10;
val b = 20;
fun f(x, y) =
    let val a = 5;
        val c = 3;
    in
        let val x = a + 1;
            val b = 2 * c;
            val a = let val c = y
                    in c + 1
                    end
        in
            [a, b, c]
        end
    end;
val answer = f(a, b);
```

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. You also may not use `if-then-else`, `let`, or pattern matching in your solutions.

a) Define a function `largest` that takes a list of lists of integers as an argument and that returns the very largest integer value to appear in any of the lists. For example, `largest([[1, 7, 2], [4], [9, 6, 5], [3, 8]])` should return `9`. You may assume the list passed contains at least one inner list element, and that each inner list contains at least one element.

b) Define a function `totalLength` that takes a list of strings as an argument and that returns the total number of characters occupied by all the strings. You may assume the list has at least one string. For example:

- `totalLength(["hi", "h o w", "ARE", "", "You?"])` should return `14`

5. Functions

Write a function `orderPairs` that accepts a list of (int, int) tuples as its parameter and that produces a list containing the same tuples, but making sure that the smaller of the two values in each tuple comes before the larger value. For example, if one of the tuples in the list is (7, 4), the corresponding tuple in your result should be (4, 7). If the tuple had been (8, 11) or (5, 5), your result would include that tuple without modification. For example, given the list:

```
val L = [(1,8),(7,2),(5,5),(4,0),(24,18),(99,~3),(55,101),(~7,~4)];
```

The call of `orderPairs(L)` would produce the following result (changed tuples are shown in bold for convenience):

```
[(1,8),(2,7),(5,5),(0,4),(18,24),(~3,99),(55,101),(~7,~4)]
```

6. Functions

Write a function `startsWith` that accepts two lists *L1* and *L2* as parameters and produces `true` if and only if *L1* begins with all of the elements of *L2* in the same order. If *L2* contains no elements, your function should produce `true` regardless of the contents of *L1*. The lists could contain any equality type as their elements. For example:

- `startsWith([1,8,2,7,5,5,0,4], [1,8,2,7])` should return `true`
- `startsWith([6,2,2,9,3], [6,2,2,9,3])` should return `true`
- `startsWith([42], [])` should return `true`
- `startsWith([], [42])` should return `false`
- `startsWith([1,8,2,7,5,5,0,4], [8,2,1])` should return `false`
- `startsWith(["a", "bc", "def"], ["bc", "def"])` should return `false`

For full credit, your function should run in no worse than O(*mn*) time, where *mn* is the product of the lengths of *L1* and *L2*. (This doesn't mean that you are never allowed to look at an element twice; it just means you shouldn't make *n* passes over the entire list *L1*, etc.) You may not call the library functions `List.take`, `List.nth`, or `List.exists`; but otherwise you may use any ML constructs you like.

7. Functions

Write a function `mapAll` that accepts a list of functions *F* and a list of values *V* as parameters and produces a new list where all of the given functions in *F* have been applied to all of the values in *V* in their given original order. Each function in *F* accepts as a parameter one value of the same type as the elements of *V*, and returns the same type. If *F* is an empty list, your function should produce *V* unmodified. For example, given the following definitions:

```
        (*   square      mult. by 2     add 1   *)
val F = [fn(x) => x*x, curry op* 2, curry op+ 1];
val V = [3,1,0,4,6];
```

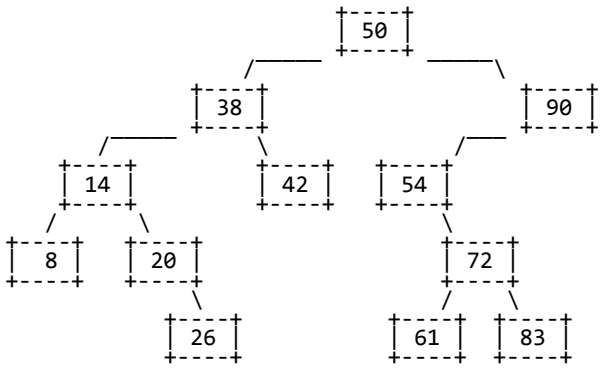
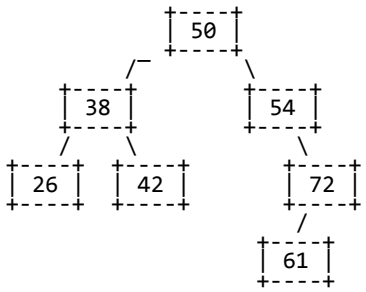
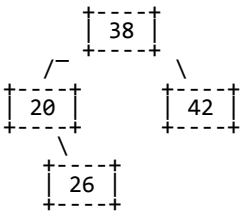
For the above definitions, the call of `mapAll(F, V)` would return a new list containing each element of `V` squared, then multiplied by 2, then added to 1 (since that happens to be what the functions in `F` do in this example). The result returned from this call would be `[19,3,1,33,73]`. You may use any ML constructs you like.

8. Datatypes

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

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

Write a function `prune` that accepts an `IntTree` *t*, an int *min*, and an int *max* as parameters and that returns a new tree consisting of the elements of *t* that are between *min* and *max* inclusive. The new tree should retain the same overall shape, though pruning the tree might change its root and other nodes. For example, given the initial tree *t* shown below at left, the table columns to the right show two trimmed versions of *t* based on calls to your function:

t	**prune(t, 25, 72)**	**prune(t, 18, 42)**
		

9. Functions

Write a function `indexOf` that accepts two lists of integers *listA* and *listB* as parameters and returns the index of the start of the first occurrence of *listB*'s elements in *listA*, or -1 (`~1`) if *listA* does not contain *listB*. If *listB* occurs more than once in listA, return the index of the first occurrence. The elements of *listB* must appear in *listA* consecutively and in the same order. Any *listA* is considered to contain the empty list at index 0.

For example, consider the following list (a few elements are bolded to correspond to calls below):

```
(*  index    0  1  2  3  4   5  6  7  8  9  10 11 12 13 14 15 16 *)
val listA = [1, 5, 7, 3, 8, ~2, 3, 4, 5, 8, ~2, 3, 2, 5, 6, 2, 7];
```

- `indexOf(listA, [8, ~2, 3])` should return `4`
- `indexOf(listA, [3])` should return `3`
- `indexOf(listA, listA)` and `indexOf(listA, [])` should return `0`
- `indexOf(listA, [1, 5, 3])` should return `~1`
- `indexOf([4, 6], [5, 2, 4, 6])` should return `~1`
- `indexOf([], [42])` should return `~1`

For full credit, your function should run in no worse than O(*mn*) time, where *mn* is the product of the lengths of *listA* and *listB*. You may use any ML constructs, unless there's a `List.indexOf` function; if there is, don't use that.