CSE 341 Sample Midterm #2

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 words = ["live", "long", "and", "prosper"];
```

Expression	Value
a) `reduce(op *, 3--5)`	_____
b) `mapx(hd o tl, [8--12, 4--9, 7--8, 8--10])`	_____
c) `mapx(fn x  => x mod 4, 1--7)`	_____
d) `reduce(op +, mapx(fn x => x * x, 1--4))`	_____
e) `filterx(fn x => x mod 4 > 1, 1--12)`	_____
f) `mapx(fn x => "(" ^ x ^ ")", words)`	_____
g) `reduce(op @, [1--3, 4--6, 5--7])`	_____
h) `mapx(fn x => real(x) + 0.5, 1--4)`	_____
i) `reduce(op +, mapx(size, words))`	_____
j) `reduce(op *, mapx(fn x => 2 * x - 1, 2--4))`	_____

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 = [("foo", 3), ("bar", 2), ("baz", 19)];
```

Expression	Type
a) `lst`	_____
b) `[tl(lst)]`	_____
c) `(8, [["a", "b"], ["c", "d"]])`	_____
d) `fn x => (hd(x) mod 3, tl(x))`	_____
e) `real o hd o tl`	_____

3. Scope

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

```
val y = 10;
val z = 20;
fun f(n, z) =
    let val x = 4 + y
        val y =
            let val x = 2
                val z = 3
            in  n + x + z
            end;
    in  x + y + n + z
    end;
val answer = f(40, 100);
```

CSE 341 Sample Midterm #2

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 `sumPositives` that takes a list of integers as an argument and that returns the sum of the positive values in the list.

b) Define a function `strip` that takes a list of nonempty strings as an argument and that returns the list obtained by stripping the last character from each string. For example:

- `strip(["hello", "there", "old", "pal"])` should return `["hell","ther","ol","pa"]`

CSE 341 Sample Midterm #2

5. Functions

Write a function `powers` that takes integer arguments n and m and that returns a list of the powers of m from 0 to n inclusive. For example, `powers(5, 2)` should return `[1,2,4,8,16,32]` (which is 2^0, 2^1, ..., up to 2^5). Your function must run in O(n) time, which means that it should require approximately n multiplications. You may assume that n is not negative.

6. Functions

Write a function `sortedChars` that takes a string s as an argument and that returns the string composed of the sorted lowercase version of the alphabetic characters from s. For example, `sortedChars("Stuart Reges??")` should return `"aeegrrssttu"`. This function would be useful for finding anagrams. For example, if you make the call `sortedChars("Sugar--Street")`, you get the same result, indicating that these two strings are anagrams. In writing your function, you may call `Char.isAlpha` to test whether a character is a letter and `Char.toLower` to convert a letter to lowercase.

CSE 341 Sample Midterm #2

7. Functions

Write a function `interleave` that takes two lists as arguments and that returns the list obtained by combining elements from the two lists in an alternating fashion. The first pair of values in the result should be the first values of the two lists. The second pair of values in the result should be the second values in the two lists. And so on. In each pair, the first value should be from the first list passed as a parameter and the second value should be from the second list passed as a parameter (see first two example calls below). If one list has more values than the other, then those values should be appended after the interleaved pairs (see last two example calls below).

- `interleave([1, 2, 3], [10, 20, 30])` should return `[1,10,2,20,3,30]`
- `interleave([10, 20, 30], [1, 2, 3])` should return `[10,1,20,2,30,3]`
- `interleave([1, 2], [10, 20, 30, 40, 50])` should return `[1,10,2,20,30,40,50]`
- `interleave([1, 2, 3, 4, 5], [10, 20])` should return `[1,10,2,20,3,4,5]`

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;
```

a) Write a function `nodes` that takes an `IntTree` as a parameter and returns the number of nodes in the tree.

b) Write a function `leaves` that takes an `IntTree` as a parameter and that returns a list of the data values stored in leaf nodes of the tree. The leaf values should be listed from left to right (i.e., in the same order in which they would be visited by any standard tree traversal).

CSE 341 Sample Midterm #2

9. Functions

a) Write a function `cartesianProduct` that takes two lists as arguments and that returns a list that contains each of the ordered pairs (x, y) where x is an element of the first list and y is an element of the second list. The pairs can appear in any order in the result. For example:

- `cartesianProduct([1, 2, 3], ["a", "b"])`
  could return `[(1,"a"),(1,"b"),(2,"a"),(2,"b"),(3,"a"),(3,"b")]`, or
  could return `[(1,"a"),(2,"a"),(3,"a"),(1,"b"),(2,"b"),(3,"b")]`

These are only two possible answers because the pairs can be listed in any order in the result. If either list passed as a parameter is empty, the result should be an empty list. Your function should run in O(m * n) time where m and n are the lengths of the two lists.

b) Write a function `product` that takes two lists of integers as arguments and that returns the list of all possible products formed by multiplying a value from the first list by a value in the second list. You may assume that the lists contain no duplicates. The values may appear in any order in the result. For example, the call `product([1, 2, 3], [5, 7])` should return a list of the 6 possible values obtained by multiplying one of the 3 numbers in the first list by one of the 2 numbers in the second list. One possible answer is `[5,7,10,14,15,21]`. You should use `cartesianProduct` to write your solution. As with `cartesianProduct`, if either list passed as a parameter is empty, the result should be an empty list.

10. Functions

Write a function `factors` that takes the prime factorization of an integer as an argument and that returns a list of all of its factors. Recall that a factor of an integer is a number that goes evenly into it. For example, the factors of 24 are `[1,2,3,4,6,8,12,24]`. Your function will be passed a list of tuples of the form (p, n) where each tuple indicates that the number has a factor of p to the nth power where p is a prime. For example, the prime factorization of 24 is 2^3 * 3^1, so it would be represented as `[(2, 3), (3, 1)]`. The factors can appear in any order. For example:

- `factors([(2, 3), (3, 1)])` could return `[1,2,3,4,6,8,12,24]` or `[1,3,2,6,4,12,8,24]`

Because the factors can appear in any order, these represent just two of the possible correct answers. As another example, if we wanted to compute the factors of 600:

- `factors([(2, 3), (3, 1), (5, 2)])` could return
  `[1,5,25,3,15,75,2,10,50,6,30,150,4,20,100,12,60,300,8,40,200,24,120,600]`

Your function must use the prime factorization to solve this problem. You can't, for example, search for factors by checking each consecutive integer up to the square root, as we did in a past homework assignment. Another way of saying this is that the running time of your function has to be related to the number of factors rather than the magnitude of the factors.

CSE 341 Sample Midterm #2
Solutions

1. Expressions

Expression	Value
a) `reduce(op *, 3--5)`	`60`
b) `mapx(hd o tl, [8--12, 4--9, 7--8, 8--10])`	`[9,5,8,9]`
c) `mapx(fn x  => x mod 4, 1--7)`	`[1,2,3,0,1,2,3]`
d) `reduce(op +, mapx(fn x => x * x, 1--4))`	`30`
e) `filterx(fn x => x mod 4 > 1, 1--12)`	`[2,3,6,7,10,11]`
f) `mapx(fn x => "(" ^ x ^ ")", words)`	`["(live)","(long)", "(and)","(prosper)"]`
g) `reduce(op @, [1--3, 4--6, 5--7])`	`[1,2,3,4,5,6,5,6,7]`
h) `mapx(fn x => real(x) + 0.5, 1--4)`	`[1.5,2.5,3.5,4.5]`
i) `reduce(op +, mapx(size, words))`	`18`
j) `reduce(op *, mapx(fn x => 2 * x - 1, 2--4))`	`105`

2. Types

Expression	Type
a) `lst`	`(string * int) list`
b) `[tl(lst)]`	`(string * int) list list`
c) `(8, [["a", "b"], ["c", "d"]])`	`int * string list list`
d) `fn x => (hd(x) mod 3, tl(x))`	`int list ->  int * int list`
e) `real o hd o tl`	`int list -> real`

3. Scope

```
val answer = 199
```

4. Curried Functions

```
val sumPositives = reduce2 op+ o (filter2 (curry op< 0));
val strip = map2 (implode o rev o tl o rev o explode);
```

5. Functions

```
fun powers(n, m) =
    let fun loop(0, product) = [product]
        |   loop(i, product) = product::loop(i - 1, product * m)
    in loop(n, 1)
    end;
powers(5, 2);
```

6. Functions

```
fun sortedChars(s) = implode(quicksort(op <=, mapx(Char.toLower,
                          filterx(Char.isAlpha, explode(s)))));
```

CSE 341 Sample Midterm #2
Solutions (continued)

7. Functions

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

8. Data Types

```
a)
fun nodes(Empty) = 0
|   nodes(Node(_, left, right)) = 1 + nodes(left) + nodes(right);

b)
fun leaves(Empty) = []
|   leaves(Node(root, Empty, Empty)) = [root]
|   leaves(Node(_, left, right)) = leaves(left) @ leaves(right);
```

9. Functions

```
a)
fun cartesianProduct([], _) = []
|   cartesianProduct(x::xs, ys) =
        map(fn y => (x, y), ys) @ cartesianProduct(xs, ys);

b)
fun product(xs, ys) = map(fn (x, y) => x * y, cartesianProduct(xs, ys));
```

10. Functions

```
fun factors([(p, n)]) = powers(n, p)
|   factors((p, n)::rest) = product(powers(n, p), factors(rest));
```