# CSE 341 Sample Midterm #3

**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 lst = [1--2, 2--4, 17--17, 7--9, 3--6];
```

| Expression | Value |
|---|---|
| `a) reduce(op *, 4--6)` | _____ |
| `b) mapx(hd, lst)` | _____ |
| `c) reduce(op +, mapx(length, mapx(tl, lst)))` | _____ |
| `d) mapx(fn(x) => reduce(op *, x), lst)` | _____ |
| `e) reduce(op @, [1--2, 2--3, 3--4])` | _____ |
| `f) filterx(fn(x) => x mod 3 = 1, 1--20)` | _____ |
| `g) mapx(fn(x) => reduce(op *, 3--x), 4--6)` | _____ |
| `h) reduce(op ^, ["cs", "is", "fun"])` | _____ |
| `i) mapx(fn(x) => (x, 2 * x), 3--6)` | _____ |
| `j) filterx(fn(x, y) => x < y, [(1, 1), (2, 3), (1, 4), (5, 4), (2, 3)])` | _____ |

**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 = [17, 9, 4, 8, 12];
```

| Expression | Type |
|---|---|
| `a) lst` | _____ |
| `b) (tl(lst), hd(lst));` | _____ |
| `c) fn x => round(real(x) * 1.5);` | _____ |
| `d) abs o hd o hd;` | _____ |
| `e) fn (x, y) => x @ [y];` | _____ |

**3. Scope**

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

```
val n = 2;
fun f(x) =
    let val y =
            let val y = x + n
                val n = 20
            in x + y + n
            end
        val x = 30
    in x + y + n
    end
val n = 4;
val answer = f(10);
```

# CSE 341 Sample Midterm #3

**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 `f1` that takes a list of integers as an argument and that returns the number of integers in the list that are greater than 7.

**b)** Define a function `f2` that takes a character as an argument and that returns the character with ordinal value one higher (e.g., f2(#"a") returns #"b").

**c)** Define a function `f3` that returns the maximum value of a nonempty list of integers (remember that you may not use `Int.max`).

**5. Functions**

Write a function `isMagnitudeSorted` that takes a list of integers and that returns `true` if and only if the values in the list are sorted in non-decreasing order by magnitude (i.e., absolute value). By definition, an empty list and a list of length one are sorted.

**6. Functions**

Implement the list function `nth` that takes a list and an integer position as arguments and that returns the value at the given position in the list. As in Java, we use zero-based indexing so that the first element of the list has index 0. For example:

- `nth([1, 8, 7, 9, 12, 15], 2)` returns 7
- `nth(["hello", "how", "are", "you"], 1)` returns "how"

You may assume that the index is a legal position in the list, which means that you may also assume that your function is passed a nonempty list.

# CSE 341 Sample Midterm #3

**7. Functions**

Write a function `median` that returns the median value of a list of real numbers. The median is defined as the value with the property that half of the numbers come before it numerically and half come after it numerically. If the list has odd length, there will be exactly one median value. If the list has even length, then you should return the average of the two middle values. You may assume that your function is passed a nonempty list of reals, but you may not assume that the list is in sorted order.

**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 `max` that takes an `IntTree` as a parameter and that returns the maximum value in the tree as an int option. Your function should return `NONE` if the tree is empty and otherwise should return the maximum value. Your function must take advantage of the fact that it is a binary search tree so that it can run in O(log n) time assuming the tree has n nodes and is balanced.

**b)** Write a function `countAtLevel` that takes an `IntTree` and an integer n as arguments and that returns a count of how many values are stored at level n in the tree. The overall root is considered to be at level 1, its children are at level 2, its grandchildren are at level 3, and so on. If n is 0 or less, your function should return 0.

# CSE 341 Sample Midterm #3

**9. Functions**

Write a function `rotations` that takes a list as an argument and that returns a list of rotations of the list that involve moving values from the front of the list to the back of the list. The first value in your answer should be the original list, followed by the list obtained by rotating one value from the front to the back, followed by the list obtained by rotating two values from the front to the back and so on. For example:

- `rotations(1--4)` should return `[[1,2,3,4],[2,3,4,1],[3,4,1,2],[4,1,2,3]]`

Your function should work on lists of any type. Your function should return an empty list if passed an empty list.

**10. Functions**

Write a function `inversions` that takes a list of integers as an argument and that returns a list of the inversions in the list as a list of tuples. An inversion is a pair of numbers x and y where x occurs before y in the list and x is greater than y. For example:

- `inversions([5, 7, 6, 8, 3, 19, 2, 24])`
  should return `[(5,3),(5,2),(7,6),(7,3),(7,2),(6,3),(6,2),(8,3),(8,2),(3,2),(19,2)]`

The value 5 has two inversions because later in the list we find the values 3 and 2, which are less than 5. Similarly the value 7 has three inversions because later in the list we find the values 6, 3 and 2 which are less than 7. The inversions may appear in any order in your answer, so the answer above is just one possible correct answer.

The list may contain duplicates, as in:

- `inversions([5, 3, 7, 3, 1, 9, 3, 12])`
  should return `[(5,3),(5,3),(5,1),(5,3),(3,1),(7,3),(7,1),(7,3),(3,1),(9,3)]`

Notice that each duplicate can generate an inversion, as in the 3 occurrences of (5,3) and the two occurrences of (3,1).

# CSE 341 Sample Midterm #3
## Solutions

**1. Expressions**

| Expression | Value |
|---|---|
| a) `reduce(op *, 4--6)` | `120` |
| b) `mapx(hd, lst)` | `[1,2,17,7,3]` |
| c) `reduce(op +, mapx(length, mapx(tl, lst)))` | `8` |
| d) `mapx(fn(x) => reduce(op *, x), lst)` | `[2,24,17,504,360]` |
| e) `reduce(op @, [1--2, 2--3, 3--4])` | `[1,2,2,3,3,4]` |
| f) `filterx(fn(x) => x mod 3 = 1, 1--20)` | `[1,4,7,10,13,16,19]` |
| g) `mapx(fn(x) => reduce(op *, 3--x), 4--6)` | `[12,60,360]` |
| h) `reduce(op ^, ["cs", "is", "fun"])` | `"csisfun"` |
| i) `mapx(fn(x) => (x, 2 * x), 3--6)` | `[(3,6),(4,8),(5,10),(6,12)]` |
| j) `filterx(fn(x, y) => x < y, [(1, 1), (2, 3), (1, 4), (5, 4), (2, 3)])` | `[(2,3),(1,4),(2,3)]` |

**2. Types**

| Expression | Type |
|---|---|
| a) `lst` | `int list` |
| b) `(tl(lst), hd(lst));` | `int list * int` |
| c) `fn x => round(real(x) * 1.5);` | `int -> int` |
| d) `abs o hd o hd;` | `int list list -> int` |
| e) `fn (x, y) => x @ [y];` | `'a list * 'a -> 'a list` |

**3. Scope**

```
val answer = 74
```

**4. Curried Functions**

```
val f1 = length o (filter2 (curry op< 7));
val f2 = chr o (curry op+ 1) o ord;
val f3 = hd o (curry quicksort op>=)
```

**5. Functions**

```
fun isMagnitudeSorted([]) = true
|   isMagnitudeSorted([x]) = true
|   isMagnitudeSorted(a::b::rest) =
        abs(a) <= abs(b) andalso isMagnitudeSorted(b::rest);
```

**6. Functions**

```
fun nth(a::_, 0) = a
|   nth(_::rest, n) = nth(rest, n - 1);
```

# CSE 341 Sample Midterm #3
## Solutions (continued)

**7. Functions**

```
fun median(lst) =
    let val n = length(lst)
        val half = n div 2;
        val lst2 = quicksort(op <, lst)
    in
        if n mod 2 = 1 then nth(lst2, half)
    else (nth(lst2, half - 1) + nth(lst2, half)) / 2.0
    end;
```

**8. Data Types**

```
a)
fun max(Empty) = NONE
|   max(Node(data, _, Empty)) = SOME data
|   max(Node(_, _, right)) = max(right);

b)
fun countAtLevel(Empty, _) = 0
|   countAtLevel(_, 1) = 1
|   countAtLevel(Node(data, left, right), n) = countAtLevel(left, n - 1)
                                              + countAtLevel(right, n - 1);
```

**9. Functions**

```
fun rotations([]) = []
|   rotations(lst) =
    let fun process(x::xs, n) =
        if n = 0 then []
        else (x::xs)::process(xs @ [x], n - 1)
    in process(lst, length(lst))
    end;
```

**10. Functions**

```
fun inversions([]) = []
|   inversions(a::rest) =
         mapx(fn(x) => (a, x), filterx(fn(x) => x < a, rest)) @ inversions(rest);
```