CSE 341 Sample Final Exam Problem Set #1

(The real exam won't have this many problems. We are giving you longer practice problem sets so you will have lots of problems to practice when you study. The kinds of problems you see will match the ones shown on these practice tests on our web site.)

1. Big Picture (short answer)

What are two significant differences, and one significant similarity, between the designs of ML and Scheme? Talk about bigger concepts and not minor syntax details. For each item, describe what it is in one sentence, and then briefly mention how it differs or is similar between the two languages.

2. ML Functions

Define a ML function called `triples` that accepts a list of any type and reverses the order of each group of three successive elements. For example, the elements a, b, c would become c, b, a. If the list's length is not evenly divisible by 3, leave the order of the excess elements unmodified. For example:

```
- triples ["a", "b", "c", "d", "e", "f"];
val it = ["c","b","a","f","e","d"] : string list
- triples [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110];
val it = [30,20,10,60,50,40,90,80,70,100,110] : int list
```

3. ML Curried Functions / Composition

Define a ML function called `maxOdd` that accepts a list of integers and returns the maximum odd integer. Define the function using a `val` declaration with curried functions and the function composition operator `o`. Do not define any helper functions using `fun` declarations or the `fn` anonymous function notation. You may assume that all of the values in the list are non-negative and that the list contains at least one odd value.

```
- maxOdd [1, 10, 9, 7, 8, 2, 14];
val it = 9 : int
```

CSE 341 Sample Final Exam Problem Set #1

4. Scheme Expressions

For each sequence of Scheme expressions below, indicate what value the final expression evaluates to. If an expression produces an error when evaluated, write `error` as your answer. Be sure to write a value of the appropriate type (e.g., `7.0` rather than `7` for a `real`; `String`s in quotes, e.g. `"hello"`; symbols with a leading quote, e.g. `'hello` ; `true` or `false` for a `bool`).

Expression — Value

a)
```
(define a 101)
(define b 103)
(define c 105)
(let ((c (+ a c))
      (a (+ b c)))
  (list a b c))
```

b)
```
(define a 101)
(define b 103)
(define c 105)
(let* ((b (+ a b))
       (a (+ b c)))
  (list a b c))
```

c)
```
(define x 101)
(define (f n) (+ x y n))
(define y 103)
(define a (f 2))
(define x 105)
(define b (f 2))
(define c a)
(set! a 19)
(list a b c)
```

d)
```
(define a '(1 2 3))
(define b '(2 3))
(define c (append (cdr b) a))
(define d (list a b))
(list c (eq? a (car d)) (eq? (cdr a) b) (eq? (cdr c) a))
```

5. Scheme

Define a Scheme procedure called `occurrences` that takes a value and a list as arguments and that returns the number of occurrences of the value in the list. Your procedure should use a deep equality comparison, although it should not search for embedded occurrences (those inside a sublist). You may assume that the second argument is a list. For example:

```
> (occurrences 3 '(7 9 2 4 a (3 2) 3 "hello" 3))
2
> (occurrences 'a '(3 a b a 19 (a b) c a))
3
> (occurrences '(a b) '(a b c (a b) d 3 a b (a b) 7))
2
> (occurrences 2 '(1 (2 3) 4 5 (2 2 2)))
0
```

CSE 341 Sample Final Exam Problem Set #1

6. Scheme

Define a Scheme procedure called `remove-all` that takes a value and a list as arguments and that returns the result of removing all occurrences of the value from the list. Your procedure should use a deep equality comparison, although it should not remove embedded occurrences (those inside a sublist). For example:

```
> (remove-all 2 '(1 2 3 4 2 7 2 (1 2 2) (3 (2 4) 2)))
(1 3 4 7 (1 2 2) (3 (2 4) 2))
> (remove-all 'a '(a b c (a b) a (c) a a))
(b c (a b) (c))
> (remove-all '(a 1) '(13 (a 1) 7 a 1 (a 1) 4 a 1))
(13 7 a 1 4 a 1)
> (remove-all 2 '(1 3 3 4 (2 2 2) 7 (2 1 2)))
(1 3 3 4 (2 2 2) 7 (2 1 2))
```

7. Scheme

Define a Scheme procedure called `maximum` that takes a list of numbers as an argument and that returns the largest value in the list. For example:

```
> (maximum '(1 13 -408 273 17 304 8 9))
304
> (maximum '(7.4 28 13.9 3.14159 2.71828 42))
42
```

You may assume that the value you are passed is a list and that it contains only numbers. Your procedure should call the `error` procedure with the message "max of empty list" if passed an empty list:

```
> (maximum ())
. . max of empty list
```

Your procedure must be efficient in several respects. It should be tail-recursive, it should run in O(n) time where n is the length of the list, and it should check the error condition only once. You are not allowed to call the standard procedure called `max` in solving this problem.

8. Scheme

Define a Scheme procedure called `switch-pairs` that takes a list as an argument and that returns the list obtained by switching successive pairs of values in the list. For example:

```
> (switch-pairs ())
()
> (switch-pairs '(a b))
(b a)
> (switch-pairs '(a b c d e f))
(b a d c f e)
> (switch-pairs '(3 a (4 5) (6 7) 9 (1 2 3)))
(a 3 (6 7) (4 5) (1 2 3) 9)
```

If the list has an odd number of values, then the final element should not be moved. For example:

```
> (switch-pairs '(a))
(a)
> (switch-pairs '(a b 3 (4 5) 9))
(b a (4 5) 3 9)
```

Your procedure should call the error procedure with the message "not a list" if passed something other than a list:

```
> (switch-pairs 3)
. . not a list
```

Your procedure must run in O(n) time where n is the length of the list.

9. Scheme

Define a Scheme procedure called `pattern` that returns a new value that represents the kind of data passed to it. Numbers, strings, symbols, and booleans should be converted into the symbols number, string, symbol, and boolean, respectively. For example:

```
> (pattern 13)
number
> (pattern 13.4)
number
> (pattern "hello")
string
> (pattern 'a)
symbol
> (pattern #t)
boolean
```

A list of values should be converted into the list of patterns for each value in the list, as in:

```
> (pattern '(13 13.4 "hello" a #t))
(number number string symbol boolean)
```

This conversion should occur at all levels of a list, preserving the list structure:

```
> (pattern '(13 (13.4 "hello" (a #t)) (14 (a b))))
(number (number string (symbol boolean)) (number (symbol symbol)))
> (pattern '(((((17)))))
((((number))))
```

Values that are not of one of these types (e.g., dotted pairs) should be included without modification in the result:

```
> (pattern '(3 (3 . 4) a "hello" (a . b)))
(number (3 . 4) symbol string (a . b))
```

Remember that Scheme has predicates for testing whether data is of a certain type (`number?`, `string?`, `symbol?`, `boolean?`, `list?`, etc.).

10. Scheme

Define a Scheme procedure called `overlap` that takes two sorted lists of numbers as arguments and that returns a list of the values in common between the two lists. For example:

```
> (overlap '(1 2 2 3 3 4 7 7 7) '(1 2 2 2 2 3 5 7 8))
(1 2 2 3 7)
```

You may assume that your procedure is passed two lists, that they contain only numbers, and that the numbers appear in sorted (nondecreasing) order. But as in the example above, they might contain duplicates. Any given number from a list can match at most once. Thus, the two occurrences of 2 in the first list can match 2 of the occurrences of 2 in the second list, but the additional occurrences of 2 in the second list do not appear in the result because they aren't matched. Similarly, the result includes only one 7 even though the first list has three occurrences of 7 because the second list has only 1 such occurrence.

Your procedure must take advantage of the fact that the lists are sorted. In particular, your procedure has to run in $O(n + m)$ time where n is the length of the first list and m is the length of the second list.

CSE 341 Sample Final Exam Problem Set #1

11. Scheme

Define a Scheme procedure called `histogram` that takes a list of values as an argument and that returns a list of 2-element lists where each 2-element list contains a value from the original list followed by a count of that value in the original list. Each value from the original list should produce exactly once such pair and they should appear in the order in which the values first appear in the original list. For example:

```
> (histogram '(1 3 2 5 1 3 3 3))
((1 2) (3 4) (2 1) (5 1))
```

The result indicates that the value 1 appeared twice in the original list, the value 3 appeared 4 times, the value 2 appeared 1 time, and the value 5 appeared 1 time.

You may assume that your procedure is passed a list, but the list will not necessarily contain simple values like numbers:

```
> (histogram '(a b c (a b) c (a b) b c a (a b)))
((a 2) (b 2) (c 3) ((a b) 3))
```

12. JavaScript Expressions

Assuming that the following variables have been defined:

```
var x = [10, 11, 12, 13, 14];
var y = {b: "a", z: x, 10: 3, length: 10};
```

Part A: What output is produced, and what value is returned, by the following code?

```
x.map(function(y) { print(y + 2); });
```

Part B: What value is returned by the following expression?

```
x.filter(function(n) { return n % 2 == 0; });
```

Part C: What output is produced by the following code?

```
print(y.b, y.length, y["z"].length, y[y.length]);
```

13. JavaScript

Extend all JavaScript arrays to have a new method called `switchPairs` that returns a new array that contains the same values as the current array, but with successive pairs of values switched in order. If the array has an odd number of values, then the final element should not be moved. For example:

```
> [].switchPairs()
[]
> ["a"].switchPairs()
["a"]
> switch_pairs(["a", "b", "c", "d"])
["b", "a", "d", "c"]
> ["a", "b", 3, [4, 5], 9].switchPairs()
["b", "a", [4, 5], 3, 9]
> [1, 2, 3, 4, "a", "b", [3, 4], [5, 6]].switchPairs()
[2, 1, 4, 3, "b", "a", [5, 6], [3, 4]]
```

Your method should run in O(*n*) time where *n* is the length of the array. Your method should not modify the array that it is called upon.

CSE 341 Sample Final Exam Problem Set #1

14.

Write a JavaScript function called `letterCount` that accepts a string parameter *s* and returns an object "map" that represents a set of counters of all the letters in the alphabet from a to z. The returned object's property names are keys for the letters of the alphabet, a-z, in uppercase, for whichever letters appear in the string *s*; and the values associated with each key are the number of occurrences of each letter in the string *s*. The object should also contain a property named `length` whose value is the combined count of letters in the string *s* (ignoring any other non-letter characters that appear in *s*). For example, the call of `letterCount("Hello, DOLLY!!")` should return an object like this:

```
{D: 1, E: 1, H: 1, L: 4, O: 2, Y: 1, length: 10}
```

Your method should run in O(*n*) time where *n* is the length of the string.

15.

Extend all JavaScript arrays to have a new method called `pairwise` that accepts a function representing a 2-argument boolean predicate. Your `pairwise` method should return `true` (or any truthy value) if the predicate returns `true` (or any truthy value) when called on each adjacent pair of values in the array; you should return `false` (or any falsy value) otherwise. For example, we can use this method to test whether an array is sorted as follows:

```
> [3, 18, 24, 24, 42, 50, 75].pairwise(function(a, b) { return a <= b; })
true
```

The method compares the first adjacent pair (3, 18) to make sure that the predicate is true, then compares the second adjacent pair (18, 24) to make sure that the predicate is true, then compares the third adjacent pair (24, 24), and so on, until it compares the final adjacent pair (50, 75). Below is an example of a predicate that tests whether a list is composed of consecutive values:

```
> [3, 18, 24, 24, 42, 50, 75].pairwise(function(a, b) { return a + 1 == b; })
false
> [3, 4, 5, 6, 7, 8].pairwise(function(a, b) { return a + 1 == b; })
true
> [3, 4, 5, 6, 7, 8, 10].pairwise(function(a, b) { return a + 1 == b; })
false
```

In the last case above, the only adjacent pair that fails is (8, 10). The method should return `true` (or any truthy value) if there are no adjacent pairs in the array to compare.

Your method should run in O(*n*) time where *n* is the length of the array and should stop checking as soon as it finds a pair that fails. Your method should not modify the original array it is called upon.

CSE 341 Sample Final Exam Problem Set #1

16. JavaScript

Define a JavaScript function called `mode` that accepts an array of integers as a parameter and returns the most commonly occurring element in the array (statisticians call this value the *mode*). Assume that values are arranged in ascending order so that duplicates appear next to each other in the array. If the elements are not in ascending order, or if the array is empty, your method should return `null`. Otherwise your method should return an object with a `value` property that stores the mode and a `count` property that stores the number of occurrences of the mode.

For example, the call of `mode([1, 8, 8, 8, 10, 14, 14, 25, 25])` should return an object containing `{value: 8, count: 3}`. This result indicates that 8 occurred most frequently and that it occurred 3 times.

If there is a tie, your method should return the first value. For example, the call of:

```
mode([6, 7, 7, 7, 7, 10, 15, 15, 15, 15, 29, 29, 29, 29])
```

should return an object containing `{value: 7, count: 4}`. Three different values occurred 4 times each (7, 15, and 29), but the method returned the first one.

The method should return `null` if called on an empty array or if the array is not in sorted order:

```
> mode([])
null
> mode([4, 4, 2, 1, 19, 5, 7])
null
```

Your method must run in O(*n*) time where *n* is the length of the array. Your method should not modify the array.

CSE 341 Sample Final Exam Problem Set #1
Solutions

1.

Both ML and Scheme are functional languages that have higher-order first class functions, lexical scoping, closures, and anonymous functions (lambdas). ML is statically typed, meaning that type-checking occurs early at compile time, while Scheme is dynamically typed, where type checks occur late as the code runs. ML supports a rich pattern-matching system and curried functions, while Scheme supports delayed evaluation and has too many parentheses (but its consistent s-expression syntax helps keep its language grammar very small). Scheme's syntax can be easily extended with macros; ML allows you to extend its type system via structures and signatures, and to define new operators.

2.

```
fun triples(a :: b :: c :: rest) = c :: b :: a :: triples(rest)
|   triples(L) = L;
```

3.

```
val maxOdd = List.foldl Int.max 0 o List.filter (fn x => x mod 2 = 1);
```

4.

```
a)  (208 103 206)
b)  (309 204 105)
c)  (19 210 206)
d)  ((3 1 2 3) #t #f #t)
```

5.

```
(define (occurrences v lst)
  (length (filter (lambda (n) (equal? n v)) lst)))
```

6.

```
(define (remove-all v lst)
  (filter (lambda (n) (not (equal? n v))) lst))
```

7.

```
(define (maximum lst)
  (define (explore max lst)
    (cond ((null? lst) max)
          ((> (car lst) max) (explore (car lst) (cdr lst)))
          (else (explore max (cdr lst)))))
  (if (null? lst)
      (error "max of empty list")
      (explore (car lst) (cdr lst))))
```

8.

```
(define (switch-pairs lst)
(cond ((not (list? lst)) (error "not a list"))
  ((null? lst) lst)
  ((null? (cdr lst)) lst)
  (else (cons (cadr lst) (cons (car lst) (switch-pairs (cddr lst)))))))
```

CSE 341 Sample Final Exam Problem Set #1

9.

```
(define (pattern item)
  (cond ((number? item) 'number)
        ((string? item) 'string)
        ((symbol? item) 'symbol)
        ((boolean? item) 'boolean)
        ((list? item) (map pattern item))
        (else item)))
```

10.

```
(define (overlap lst1 lst2)
  (cond ((null? lst1) ())
        ((null? lst2) ())
        ((< (car lst1) (car lst2)) (overlap (cdr lst1) lst2))
        ((> (car lst1) (car lst2)) (overlap lst1 (cdr lst2)))
        (else (cons (car lst1) (overlap (cdr lst1) (cdr lst2))))))
```

11.

```
(define (histogram lst)
  (if (null? lst)
      lst
      (let ((v (car lst)))
        (cons (list v (occurrences v lst))
            (histogram (remove-all v lst))))))
```

12.

```
a)  output:  12
             13
             14
             15
             16
    returns: [undefined, undefined, undefined, undefined, undefined]

b)  returns: [10, 12, 14]
c)  output:  a 10 5 3
```

13.

```
Array.prototype.switchedPairs = function() {
    var result = [];
    for (var i = 0; i < this.length; i += 2) {
        result.push(this[i + 1]);
        result.push(this[i]);
    }
    if (this.length % 2 != 0) {
        result.push(this[this.length - 1]);
    }
    return result;
};
```

CSE 341 Sample Final Exam Problem Set #1

14.

```
function letterCount(s) {
    s = s.toUpperCase();
    var map = {length: 0};
    for (var i = 0; i < s.length; i++) {
        var c = s[i];
        if (c >= "A" && c <= "Z") {    // if (c.match(/[A-Z]/))   or
            if (map[c]) {              // if ("ABCDEFGHIJKLMNOPQRSTUVWXYZ".indexOf(c) >= 0)
                map[c]++;
            } else {
                map[c] = 1;
            }
            map.length++;
        }
    }
    return map;
}
```

15.

```
Array.prototype.pairwise = function(f) {
    for (var i = 0; i < this.length - 1; i++) {
        if (!f(this[i], this[i + 1])) {
            return false;
        }
    }
    return true;
};
```

16.

```
function mode(a) {
    if (a.length == 0) {
        return null;
    }

    var result = {count: 1, value: a[0]};
    var thisCount = 1;
    var maxCount = 1;

    for (var i = 1; i < a.length; i++) {
        if (a[i] < a[i - 1]) {
            return null;          // not properly sorted
        }
        if (a[i] == a[i - 1]) {
            thisCount++;
            if (thisCount > maxCount) {
                result.count = thisCount;
                result.value = a[i];
            }
        } else {
            thisCount = 1;
        }
    }

    return result;
};
```