Questions on ML

Important note: In all cases code should be clearly-written and should include a brief explanation in English explaining the design of your code.

Your answer must take the form of a plaintext file including the program and a nontrivial collection of tests, which can be cut-and-pasted by your marker into the command line, to test that it works.

Consistent with the principle that *code is written for humans to read* in the first instance, and for computers to execute only in the second instance, marks will be awarded for *style and clarity and making it easy for your marker to mark your work.* If your answer's a mess, don't expect the human at the other end of this question sheet to be pleased and persevere.

A model question and answer is at the end of this document.

1. **Complex number arithmetic**

The **complex numbers** are explained here (and elsewhere):
http://www.mathsisfun.com/algebra/complex-number-multiply.html
Represent a complex integer as an element of the datatype

```
datatype cint = CI of int * int.
```

(So `CI(4,5)` represents 4+5i.)

Implement functions `cadd` and `cmult` of type `cint * cint -> cint` representing complex integer addition and multiplication.

For instance,

```
cadd(CI(1,0),CI(0,1))
```

should compute

```
CI(1,1).
```

2. **Sequence arithmetic**

An **integer sequence** is an element of

```
type intseq = int list.
```

(So `intseq` is a type alias for a list of integers.)

Implement recursive functions `seqadd` and `seqmult` of type `intseq * intseq -> intseq` that implement pointwise addition and multiplication of integer sequences.

For instance

```
seqadd([1,2,3],[~1,2,2])
```

should compute

```
[0,4,5]
```

Do *not* write error-handling code to handle the cases that sequences have different lengths.

3. **Matrices**

Matrix addition and multiplication are described here:

- addition: http://www.mathsisfun.com/algebra/matrix-introduction.html
- Multiplication (dot product): http://www.mathsisfun.com/algebra/matrix-multiplying.html

Represent integer matrices as the datatype `intmatrix = IM of intseq list`.

So a matrix is a column of rows of integers.

Write functions

- `ismatrix : intmatrix -> bool`
  This should test whether a list of lists of integers represents a matrix (so the length of each row should be equal).
- `matrixshape : intmatrix -> (int * int)`
  This should return a pair that is the number of columns, and the number of elements in any row.
- `matrixadd : intmatrix * intmatrix -> intmatrix`
  Matrix addition, which is simply pointwise addition. You may find your previous answers useful.
- `matrixmult : intmatrix * intmatrix -> intmatrix`
  Similarly for matrix multiplication.

Do *not* write error-handling code for malformed input, e.g. a column of rows of integers of different lengths, or an attempt to sum matrices of different shapes.

4. **Essay-style question**

Write an essay on the ML type system. Be clear, to-the-point, and concise. Convince your marker that you understand:

- Ad-hoc and parametric polymorphism.
- Function types.
- Pair types.
- Equality types.

- Type aliases and datatype declarations (and their differences)

Include short code-fragments (as I do when lecturing) to illustrate your observations.

(Your answer to this question must be a plaintext file, but need not cut and paste into the command line, because it is an essay.)

5. **Bonus question**

- Write a pair of functions of types
  `(('a * 'b) -> 'c) -> ('a -> ('b -> 'c))`
  and
  `('a -> 'b -> 'c) -> (('a * 'b) -> 'c)`
  and explain why this was a cool question.

6. **Seriously cool bonus question**

- Write a pair of functions of types
  `int -> ('a -> 'a)`
  and
  `('a -> 'a) -> int`
  (Hint: search for “Church numerals”.)

7. **Unmarked question**

- Implement the Tower of Hanoi as a function of type
  `unit -> (int list*int list*int list)`
- Implement Bubblesort and Quicksort in ML.

Model question M:

Write a function

```
   sumf : 'a list -> ('a -> int) -> int
```

that inputs a list `l` and a function `f : 'a -> int` and outputs
the sum of `f` applied to all the elements of `l`
(so `sumf [1,2,3] (fn x => x*x)` calculates 1*1+2*2+3*3 = 21).

Model answer:

```
(*****************************)
(* Start of answer to Question M
Write a function
   sumf : 'a list -> ('a -> int) -> int
that inputs a list l and a function f : int -> int and outputs
   the sum of f applied to all the elements of l
(so sumf [1,2,3] (fn x => x*x) calculates 1*1+2*2+3*3 = 14).
*)

fun sumf [] f = 0
(* If the list is empty then the sum of the empty list is 0 *)
  | sumf (h::t) f = (f h)+(sumf t f);
(* Otherwise calculate (f h) and proceed recursively *)

(* Test 1 (should return 14): *)
sumf [1,2,3] (fn x => x*x);

(* Test 2 (should return 0): *)
sumf [1,2,-3] (fn x => x);

(* Test 3; sum squares of a list of lists (should return true) *)
sumf [[1,2,3],[4,5,6],[7,8,9]] (fn l => sumf l (fn x => x*x))
=
sumf [1,2,3,4,5,6,7,8,9] (fn x => x*x);

(* End of answer to Question M *)
(*****************************)
```

Assuming 10 points are awarded, this answer gets:

- 4 points for being a correct, well-structured program,
- 3 points for a clear explanation, and
- 3 points for exhaustive testing.

Your marker is particularly impressed that the third and final test demonstrates understanding of polymorphism.