Sample Midterm

Instruction: read questions carefully and write your answers. Please explain your answers when necessary, and do not just write down answers only. Good Luck!

Problems	Score
1. Lexical Scope	**/16**
2. Module	**/16**
3. Type Inference	**/20**
4. Equivalence	**/13**
5. Higher-Order Function	**/15**
6. ML Programming	**/20**
Total	**/100**

Problem 1. [Lexical Scope] For each of the following programs, give the value that *ans* is bound to after evaluation.

(a) (5 points)

```
val z = 0
fun f(x) =
    let
        val y = x+1
        val x = y-1
    in
        x + y + z
    end

val z = 4
val ans = f 42
```

(b) (6 points)

(c) (5 points)

Problem 2. [Module] In this problem, suppose we have an ML structure M and signature S in this standard usage:

```
signature S =
sig
...
end

structure M :> S =
struct
...
end
```

Assume everything type-checks initially, meaning M matches S. For each of the following statements, answer "always," "sometimes," or "never", with the reason (16 points) (2 points each)

(a) If S originally contains val num: int and we comment out this line, then M will still match S.

…

Problem 3. [Type Inference] Please consider the following ML programs and infer the types of all implicitly defined variables and function bindings. Justify your answers. If you need type variables, please use the following type variables ('a, 'b, 'c, ...). During the type inference you could use temporary type variables (T1, T2, ...).

(a) (4 points)

```
Line 1: fun length xs =
Line 2: case xs of
Line 3: [] => 0
Line 4: | x::xs' => 1 + (length xs')
```

After Line 1

```
length: T1 -> T2
```

After Line 2

After Line 3

After Line 4

```
Final result:
```

(b) (8 points)

Problem 4. [Equivalence] Please write whether two programs (A/B) are equivalent (Yes or No) and justify your answer. Also, predict the result of each program output.

(a) (4 points)

<table>
<tr><td>A</td><td>B</td></tr>
<tr><td><pre>
val y = 1
fun f x = x+y
fun h () =
      let val y = 2
      in
            f
      end
fun g y = (h ()) y
g (1)
</pre></td><td><pre>
val y = 1
fun f x = x+y
fun h () =
      let val y = 3
      in
            f
      end
val g = (h())
g (1)
</pre></td></tr>
<tr><td>Output : 2</td><td>Output : 2</td></tr>
<tr><td colspan="2">Equivalence? Yes
Reasons: Because of the characteristic of the lexical scope, local variable(binding) y do not affect to the function f.</td></tr>
</table>

(b) (5 points)

(c) (4 points)

Problem 5. [Higher-Order Function] Given a datatype for integer arithmetic expression (exp) and evaluation function f (with type `int -> bool`), we wish to write a function (`true_of_some_constants`) to check whether at least m constants in an expression are satisfied (evaluated to be true) by the function f.

For your reference, we provide an example for `true_of_all_constants( f, e )` as follows,

```
datatype exp = Constant of int
      | Negate of exp
      | Add of exp * exp
      | Multiply of exp * exp

fun true_of_all_constants(f,e) =
    case e of
        Constant i => f i
      | Negate e1 => true_of_all_constants(f,e1)
      | Add(e1,e2) => true_of_all_constants(f,e1)
            andalso true_of_all_constants(f,e2)
      | Multiply(e1,e2) => true_of_all_constants(f,e1)
              andalso true_of_all_constants(f,e2)

fun all_even_exp e =
      true_of_all_constants((fn x => x mod 2 = 0),e)
```

(a) Please write a ML function, `true_of_some_constants f e m (a Curried version)`. (Hint: you may define a help function, e.g., `count_of_true_contstants( f, e )`). (10 points)

```
fun true_of_some_constants f e m =




val some_even_exp = true_of_some_constants (fn x => x mod 2 = 0)
val e = Multiply( Add( Constant(2), Constant(1) ), Constant(4) )
some_even_exp e 3 (* false *)
Some_even_exp e 2 (* true *)
```

(b) Does your implementation include recursion? Can you implement the `true_of_some_constants` by using Tail-recursion? Justify your answer. Also, please explain when the tail-recursion could be useful. (5 points)

Problem 6. [ML Programming] We will implement the tournament of Rock, Paper, Scissors game. The followings are the data types we define to implement the tournament:

```
type name = string

datatype RSP =
      ROCK
    | SCISSORS
    | PAPER

datatype 'a strategy = Cons of 'a * (unit -> 'a strategy)

datatype tournament =
      PLAYER of name * (RSP strategy ref)
    | MATCH of tournament * tournament
```

RSP strategy is a stream of RSP, which, when called, yields a pair of RSP and a thunk that will give the next item in the stream. The following variables (r, s, p, rp, etc) are example RSP strategies, and the function next retrieves one RSP value from the given RSP strategy.

```
fun onlyOne(one:RSP) = Cons(one, fn() => onlyOne(one))
fun alterTwo(one:RSP, two:RSP) = Cons(one, fn() => alterTwo(two,
one))
fun alterThree(one:RSP, two:RSP, three:RSP) = Cons(one, fn() =>
alterThree(two, three, one))

val r = onlyOne(ROCK)
val s = onlyOne(SCISSORS)
val p = onlyOne(PAPER)
val rp = alterTwo(ROCK, PAPER)
val sr = alterTwo(SCISSORS, ROCK)
val ps = alterTwo(PAPER, SCISSORS)
val srp = alterThree(SCISSORS, ROCK, PAPER)

fun next(strategyRef) =
    let val Cons(rsp, thunk) = !strategyRef in
        strategyRef := thunk();
        rsp
    end
```

(a) In this question, you are asked to implement whosWinner function, which takes a tournament as its parameter and returns the winner of the tournament. For example, for the following code,

```
val winner = whosWinner(MATCH(PLAYER("s", ref s),
           MATCH(PLAYER("rp", ref rp), PLAYER("r", ref r))));
```

whosWinner returns a player who won the tournament, that is, PLAYER("rp", ref rp) in this case. Use pattern matching (case expressions) to implement whosWinner function.

```
fun whosWinner(t) =
```

Hint: The outmost pattern matching for whosWinner has three cases; 1) when the parameter t is a player, 2) when t is a match between two players, 3) when t is match of matches. (17 points)

(b) For the following match, who is the winner? (3 points)

```
val match = MATCH(
  MATCH(PLAYER("John", ref sr), PLAYER("Steve", ref s)),
    MATCH(PLAYER("Alice", ref p),
      MATCH(PLAYER("David", ref r),
        MATCH(PLAYER("Bill", ref s), PLAYER("Emily", ref srp)))))
```