1. (10 points) **True/False**
   (a) ___ F# is statically typed
   (b) ___ Tuples are heterogeneously typed and can contain other tuples
   (c) ___ `[[1; 4]; [-1; 5]; [3; 4]; 5]` is a valid F# list
   (d) ___ `let f x = (fun y -> x * y);;` is a curried function
   (e) ___ A function with one function parameter that returns a function is order 2
   (f) ___ `let f5 a b = (a+1)::b;;` can only take integers as an argument for a
   (g) ___ The `reduce` function we have written is the same as the `List.foldBack` function.
   (h) ___ The `List.fold` function is the same as `List.foldBack` for associative operations such as + and *

2. **Matching**
   (a) ___ `(fun a b -> a*b)`
   (b) ___ `(fun b -> 4*b)`
   (c) ___ `[2; 3; 4]`
   (d) ___ `12`
   (e) ___ `10`

   Match with numbers:

   1. `let f a b = a*b`
   2. `let f a = (fun b) -> a*a`
   3. `let f a b = a * b;; f 4;;`
   4. `List.foldBack (fun a b -> a*b) [3;1;4] 1`
   5. `List.map (fun a -> a+1) [1;2;3]`
   6. `List.foldBack (fun x c -> x+c) (List.map (fun a -> a+1) [1;2;3]) 1`

3. **Calculating Values**

   Use the following definitions in the upcoming questions:

```
let rec map f L =
  match L with
    | [] -> []
    | h::t -> f h::map f t;;

let rec filter f L =
  match L with
    | [] -> []
```

```
    | h::t when f h -> h::filter f t
    | h::t -> filter f t;;

let rec reduce f a L =
  match L with
    | [] -> a
    | h::t -> f h (reduce f a t);;

let rec map2 f L1 L2 =
  match (L1, L2) with
    | ([], []) -> []
    | (h1::t1, h2::t2) -> (f h1 h2)::map2 f t1 t2;;

type IS = I of int | S of string;;

let m a b =
  match (a,b) with
    | (I x, I y) -> I (x*y)
    | (I x, S y) -> S y;;

let g x y = (x, y)

let h x y = y @ [x]
```

Give the results of the following statements or write ERROR if they won't work.

(a) _____ `g 2 "b"`

(b) _____ `let f a = g a 8;; f "a";;`

(c) _____ `h [3;2] 1`

(d) _____ `map (fun a -> a+1) [1;2;3;4]`

(e) _____ `map2 g [1;2] ["a";"b"]`

(f) _____ `map2 (fun a b -> (a,b)) [1;2] ["a";"b"]`

(g) _____ `reduce h [] [1;2;3]`

(h) _____ `filter (fun x -> x > 3) [1;2;3;4;5]`

(i) _____ `m (I 5) (S "a")`

(j) _____ `m (I 5) (I 6)`

(k) _____ `m (S "a") (I 5)`

(l) _____ Domain of g

(m) _____ Range of g

4. **Programming in F#**

(a) Define a function `count5` to count the number of 5s in a list. For example:

```
count5 [] // returns 0
count5 [1;5;2;5;3;5] // returns 3
```

(b) Given a union data type definition of:

```
type irs = I of int | R of double | S of string
         | IR of int * double | IS of int * string
```

Define the `Imult` function that multiplies two I types and returns an I type. Ignore any other combinations. The function should behave like:

```
Imult (I 5) (I 4) // returns I 20
```

(c) Using the datatype definition of b, define the `IxS` function that given an I type and a S type returns an IS type. For example:

```
IxS (I 4) (S "hello") // returns IS (4, "hello")
```

(d) Given the following datatype definition:

```
type 'e mylist = NIL | CONS of 'e * 'e mylist
```

Define the function to find the length of a mylist. For example:

```
myLength (CONS (8, CONS (2, CONS (6, NIL)))) // returns 3
```

(e) Define the recursive `rdc` function that returns all but the last element of a list.

```
rdc [1;2;3] // returns [1;2]
```

(f) Define a *tail recursive* `rdc` that returns all but the last element of a list.

```
tailRdc [] [1;2;3;4] // returns [3;2;1]
//(note: this will return the list in reverse order and that is OK)
```