CSc 372, Fall 2001
ML Examination Solutions

Problem 1: (2 points each; 8 points total)

State the ***type*** of each of the following expressions, or if the expression is not valid, state why. For example, the type of the expression `3+4` is `int` and the type of the expression `length` is
`'a list -> int`.

```
(1, [2,3], 4.0)

     int * int list * real


(reduce op+)

     int list -> int


explode o rev o implode
```

Not valid – `implode` produces a string but `rev` requires an `'a list`

```
[[[size]]]

     (string -> int) list list list
```

Problem 2: (3 points each; 15 points total)

State the ***type*** of each of the following functions:

```
fun a (w,h) = w * h * 1

     int * int -> int

fun f(x) = f(1) + f(2)

     int -> int

fun f(L,(a,b)) = L = (a,b)

     (''a * ''b) * (''a * ''b) -> bool

fun f(3,4) = (size,length)

     int * int -> (string -> int) * ('a list -> int)

fun x y z = (y z) + z

     (int -> int) -> int -> int
```

Problem 3: (3 points each, 9 points total)

Edit or rewrite the following functions to make better use of the facilities of ML:

```
fun f(a,b,c) = [a-c, a+c]

      fun f(a, _, c) = [a-c, a+c];


fun f(x,y) = x::y::1::2::[]

      fun f(x,y) = [x,y,1,2];


fun f(n) = if n = 10 then true else if n = 5 then true else false;

      fun f2(10) = true
       |  f2(5)  = true
       |  f2(_)  = false
```

Problem 4: (5 points)

Write the `map` function. The type of map is `('a -> 'b) -> 'a list -> 'b list`

```
fun map F [] = []
 |  map F (x::xs) = F(x)::(map F xs)
```

Problem 5: (7 points)

Create a function `abslist(L)` of type `real list -> real list` that produces a copy of L with each value in the output list being the absolute value of the corresponding value in the input list. Assume there is NO function like Java's `Math.abs()` to compute absolute value–do the absolute value computation yourself.

```
fun abslist(L) = map (fn(x) => if x < 0.0 then ~x else x) L
```

Problem 6: (7 points)

Your instructor suffered the great embarrassment of distributing a version of `gather` that has a bug: If called with an empty list it should return `[]` but in fact it returns `[[]]`. Example:

```
- gather([], 10);
val it = [[]] : int list list
```

Change this:

```
fun gather(L, limit) =
```

to this:

```
fun gather([], _) = []
 |  gather(L, limit) =
```

Problem 7: (15 points)

In this problem you are to create TWO functions, `doubler` and `quadrupler`. `doubler` is of type `string list list -> string list list` and "doubles" each letter in the strings. `quadrupler` is of the same type, but quadruples each letter.

```
fun doubler L =
  let
    fun f([]) = []
     |  f(c::cs) = c::c::f(cs)
  in
    (map (map (implode o f o explode))) L
  end

val quadrupler = doubler o doubler
```

Problem 8a: (7 points)

Create a function `genlist` that takes a list of integers and for each integer N in the list, produces a list with N instances of the number 1. You may assume that all the values are non-negative.

```
val genlist = map ((map (fn(_) => 1)) o iota)
```

Problem 8b: (7 points)

Create a function `genlist_inv` that performs the inverse operation of `genlist`. The only value appearing in the lists will be the integer 1 (one).

```
val genlist_inv = map sum
```

An acceptable answer is to use `length` instead of `sum`,

```
val genlist_inv = map length
```

but if you try it out with the interpreter, you'll find that you get an error about type variables not being generalized.

Problem 8c: (2 points)

What is the type of `genlist_inv o genlist o genlist_inv` ?

```
int list list -> int list
```

Problem 9: (18 points)

Create a function `tacdel(fname)` that reads the file named by `fname` and prints (using the `print` function) the lines in the file in reverse order, and if a line contains the character "`@`", the line "`<D>`" appears in its place.

```
fun tacdel(fname) =
     let
          val bytes = read_all_bytes(fname)
          fun lmapper(s) =
               if member(#"@", explode s) then "<D>" else s
          val lines = map lmapper (rev (split #"\n" bytes))
    in
          print(concat(ien(lines, 1, "\n")))
    end
```