CSCI 330 Spring 1999
SECOND MIDTERM EXAM
Tuesday 99/3/99—Two Pages
Closed Book

1 ML–35 points

1. What is the type of x in the ML expression `x ^ "bc"` ? (2 points) **Answer:** string.

2. In ML, you are not required to indicate the type of all variables used in a program. Why is ML still considered a strongly typed language? (3 points) **Answer:** ML uses a sophisticated type inference algorithm to decide the type of every expression in a program from sufficient clues.

3. Write a recursive function *length* that takes a list as argument and returns the length of that list (15 points). **Answer:**

```
fun length [] = 0
|   length (x::xs) = 1 + length(xs);
```

4. For this question, you must write three versions of a recursive function *mytail* that takes a non-empty list as argument, and returns the last element of the list (25 points) .

   (a) Assume the list is non-empty. Do not use patterns. **Answer:**

   ```
   fun mytail l = if l=[x] then x else mytail(tl(l));
   ```

   (b) Assume the list is non-empty. Use patterns. **Answer:**

   ```
   fun mytail (x::nil) = x
   |   mytail (x::xs) = mytail xs;
   ```

   (c) Extend the second function you wrote by raising an exception (*Empty)*when the list is empty. **Answer:**

   ```
   fun mytail [] = raise Empty
   |   mytail (x::nil) = x
   |   mytail (x::xs) = mytail(xs);
   ```

5. Write the function *simpleMap* that applies its first argument (a function) to its second argument (a list) (15 points). **Answer:**

```
- fun simpleMap(F,[]) = []
= |   simpleMap(F,x::xs) = F(x) :: simpleMap(F,xs);
```

6. Use simpleMap to solve the following problems:

   (a) Replace every negative element of a list of reals by 0, leaving non-negative elements as they are (10 points). **Answer:**

   ```
   - fun trun x = if x >0.0 then x else 0.0;;
   val trun = fn : real -> real
   - simpleMap(trun, [2.0, ~1.0, ~5.0, 0.0, 7.52]);
   val it = [2.0,0.0,0.0,0.0,7.52] : real list
   ```

   (b) Add 1 to every element of an integer list (5 points). **Answer:**

   ```
   - - fun add1 x = x+1;
   val add1 = fn : int -> int
   - simpleMap(add1, [1,2,3]);
   ```

   (c) I claim that a Curried version of *simpleMap* would make your solution of parts (a) and (b) better. Why? **Answer:** Because you could define two partially evaluated functions that do what is required:

   ```
   fun repneg l = simpleMap trun;
   fun add1tolist l = simpleMap add1;
   ```