Name: _____

ID number: _____

CS 312, Fall 2003

Exam 2

November 18, 2003

Problem	1	2	3	4	Total
Grader					
Grade	/15	/25	/30	/30	

There are 4 problems on this exam. Please check now that you have a complete exam booklet with 8 numbered pages plus the cover sheet. *Write your name or id number on each page of the exam.* Be sure to try all of the problems, as some are more difficult than others (i.e., don't waste a lot of time on a problem that is giving you a hard time – move on to another problem and then return to it later). The order of the problems is (very roughly) by increasing difficulty.

This exam is closed book. No papers, books or notes may be used. You will be provided with a handout that gives excerpts from the Mini-ML evaluator.

There is space provided to answer each question. You may request extra paper if necessary. *Do not put any answers on the backs of pages* (THEY WILL NOT BE GRADED), ask for an extra piece of paper if you need more space.

To help ensure that you don't accidentally miss any of the questions, we have marked those sections where an answer is requested with a $\Longleftarrow$ in the right margin.

The staff reserves the right to ignore illegible answers. "Pretty printing" (proper indentation) of your code will aid in the grading process.

1. (15 points)

   Consider the following expression:

```
let
  val a = ref 5
  val b = 7
  fun marvin(n, x, y) = if n = 0
                          then x
                        else marvin(n-1, ref y, !x)
in
 SomeExpression
end
```

   (a) What is type type of `marvin`? ⟸

   (b) Draw the environment diagram for the environment in which `SomeExpression` is evaluated. Be sure to indicate ref cells with double boxes. ⟸

(c) Suppose we replace `SomeExpression` with `marvin(312,a,b)`. What is the value of the expression? ⟸

(d) Suppose we replace `SomeExpression` with `marvin(2003,a,b)`. What is the value of the expression? ⟸

2. (25 Points) Modify the evaluator so that inside each function the value of variable `argv` will be a list containing all of the arguments to the function. Note the following:

   (a) The list that `argv` refers to can contain values of different types; normal `SML` lists are type-homogeneous.

   (b) The prelim 2 version of the interpreter does not handle `fun` declarations, thus your implementation must only work for unnamed (anonymous) functions defined via `fn`.

   For example:

```
let
 val ford = fn(l:int list, n:int) => tl(argv)
in
 ford([0, 7, 14, 21],3)
end
```

   should return `[3]`. If we replaced the body of `ford` with `hd(argv)` it would return `[0,7,14,21]`. More generally, if we replaced the body of `ford` with `arthur(argv)`, then the function `arthur` would be called on the list `[3, [0, 7, 14, 21]]` (As stated, we note that such a list is not legal in `SML`, but we can handle it in `Mini-ML`.)

   The user is allowed to hide the default definition of `argv` by explicitly binding the variable. (You can assume that there are no functions with parameters named `argv`). So for example the value of

```
let
 val ford = fn(n:int) =>
              let
                val argv = hd(argv)+4
              in
                argv*(argv+1)
              end
in
 ford(1+1)
end
```

   should be (of course!) 42. For full credit, your solution should be as short and simple as possible. ⇐

3. (30 points)

Consider the following modification to the evaluator.

```
(* Original code *)
  fun lookupBinding (Env(e)) (id:string): value option =
    case List.find (fn (s, _) => id = s) e of
      NONE       => NONE
    | SOME(_, v) => SOME v
(* Modified code *)
  fun lookupBinding (Env(e)) (id:string): value option =
    case List.find (fn (s, _) => id = s) (List.rev e) of
      NONE       => NONE
    | SOME(_, v) => SOME v
```

The only change we have made is to insert a call to `List.rev`, which reverses a list.

(a) Provide an English language description of what this modification does. Keep your explanation as simple and precise as you can. ⟸

(b) Give an expression that gives two different values before and after the modification, and give those two different values. ⟸

4. (30 points)

Consider the following definitions for streams:

```
datatype 'a stream = NIL | Stream of 'a * (unit -> 'a stream)
fun intsfrom(n:int):int stream = Stream(n,fn() => intsfrom(n+1))
val Z = intsfrom(0)
(* Make a list from the first n elements of s *)
fun takeN(s: 'a stream, n: int): 'a list =
case (s, n) of
  (_, 0) => []
| (NIL, _) => raise Fail "empty stream"
| (Stream(h, t), _) => h::(takeN(t(), n-1))
```

(a) Suppose that we evaluate the expression

```
fun ford(s: 'a stream): 'a stream =
  case s of
    NIL => NIL
  | Stream(h, t) =>
        Stream(h, fn () =>
             Stream(h, fn()=>ford(t())))
```

After evaluating this definition, what is the value of the expression `takeN(ford(Z),8)`? ⟸

(b) Suppose that we evaluate the expression

```
(* Start of ford definition *)
fun ford(s: int stream, k: int): int stream =
  let
    val arthur: int ref = ref 0
    val marvin: int ref = ref k
    (* Start of zaphod definition *)
    fun zaphod(s: int stream): int stream =
      case s of
        NIL => NIL
      | Stream(h, t) => if (!marvin) < k
                        then (marvin := (!marvin) + 1;
                              Stream(!arthur, fn() => zaphod(s)))
                        else (marvin := 1;
                              arthur := h;
                              Stream(h, fn() => zaphod(t())))
    (* End of zaphod definition *)
  in
    zaphod s
  end
(* End of ford definition *)
```

After evaluating this definition, what is the value of the expression `takeN(ford(Z,3), 9)`? ⇐

(c) Suppose that we evaluate the following expressions

```
val arthur: int ref = ref 0
val marvin: int ref = ref 0
(* Start of ford definition *)
fun ford(s: int stream, k: int): int stream =
  let
    val () = marvin := k
    (* Start of zaphod definition *)
    fun zaphod(s: int stream): int stream =
      case s of
        NIL => NIL
      | Stream(h, t) => if (!marvin) < k
                        then (marvin := (!marvin) + 1;
                              Stream(!arthur, fn() => zaphod(s)))
                        else (marvin := 1;
                              arthur := h;
                              Stream(h, fn() => zaphod(t())))
    (* End of zaphod definition *)
  in
    zaphod s
  end
(* End of ford definition *)
```

After evaluating these, what is the value of the expression

```
let                                                                    ⟸
  val ZaphodsHead1 = takeN(ford(Z,3),5)
  val ZaphodsHead2 = takeN(ford(Z,3),5)
in
  ZaphodsHead1@ZaphodsHead2
end
```