Name: ______________________________

NetID: __________________________

CS 312, Fall 2003

Final Exam

December 17, 2003

DON'T PANIC

Problem	1	2	3	4	5	6	Total
Grader							
Grade	$\overline{5}$	$\overline{5}$	$\overline{15}$	$\overline{20}$	$\overline{25}$	$\overline{30}$	

There are 6 problems on this exam. Please check now that you have a complete exam booklet with 10 numbered pages plus the cover sheet. *Write your name or netid 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. (5 points)

   In this problem you will write function the function `pickKth`, which takes an integer list l and a non-negative integer k, and returns a list consisting of every kth element of l, including the first one.

   Your function should behave as follows:

```
pickKth([], 10) = []
pickKth([1, 2, 3], 4) = [1]
pickKth([9, 8, 7, 6, 5, 4, 3], 2) = [9, 7, 5, 3]
```

   (a) Give a recursive implementation of `pickKth`. $\Longleftarrow$

   (b) Is your implementation of `pickKth` tail recursive? $\Longleftarrow$

2. (5 points)

   Recall that the SML interpreter runs on a computer with 32-bit words. We can number these bits 0 through 31, where 0 is the lowest bit and 31 the highest. A field is a 32-bit word, which can represent either a direct value (such as an integer or a boolean) or a pointer. The following questions involve the details of the SML implementation discussed in lecture.

   (a) Which bit determines whether a field holds a direct value? What is the value of this bit if the word holds a direct value? $\Longleftarrow$

   (b) Recall that in the BIBOP (Big Bag of Pages) allocation scheme, main memory is divided into a small number of regions, and each region contains data of a known type. For example, there might be 8 different regions, where the first region holds integers, the second holds floating point numbers, etc. In a field which is a pointer, under this scheme, 3 bits of the pointer would determine the type of the object being pointed to. Under the standard BIBOP scheme, which 3 bits would this be? $\Longleftarrow$

3. (15 points)

   In this problem you will modify the evaluator by adding a new special form `curry` which curries takes a function and curries it. The input will be a user-defined or anonymous function. You can assume that all of the arguments to the input function are integers.

   As an example, consider the following definitions:

   ```
   val ford = fn(x:int,y:int) => x - y
   val marvin = fn(x:int,y:int,z:int) => x + (y*z)
   ```

   After you have added this special form, the value of the following three expressions will all be (surprise!) 42:

   ```
   (curry ford) 44 2
   (curry marvin) 2 10 4
   (curry (fn(x:int,y:int) => x*y)) 6 7
   ```

   To see this, note that the value of `(curry ford) 44 2` is the value of `ford(44,2)`, and similarly for the other examples. (The result of `(curry marvin)` is a depressed curried robot, which is a concept best left to the reader's imagination ...)

   List all the modifications you make to the evaluator, along with the relevant line numbers in the handout. Don't forget to add your special form to the global environment. ⟸

4. (20 points)

Consider the type definition below:

```
datatype exp = TRUE | FALSE | X | Y |
               NOT of exp | AND of exp * exp | OR of exp * exp
```

The type declaration describes all logical expressions that include boolean constants, boolean variables X and Y, and logical operators not, or, and and. For example, the expression `OR(NOT X, Y)` encodes the boolean function implication "X implies Y".

Now consider the following definitions

```
fun ford(e: exp) =
 case e of
   TRUE => (fn (x, y) => true)
 | FALSE => (fn (x, y) => false)
 | X => (fn (x, y) => x)
 | Y => (fn (x, y) => y)
 | NOT e =>
   (fn (x, y) => not ((ford e)(x, y)))
 | AND(e1, e2) =>
   (fn (x, y) => ((ford e1)(x, y)) andalso ((ford e2)(x,y)))
 | OR(e1, e2) =>
   (fn (x, y) => ((ford e1)(x, y)) orelse ((ford e2)(x, y)))

val arthur = ford(OR(NOT X, Y))
```

(a) What is the type of `arthur`? ⟸

(b) Give an expression that uses `arthur` and its value. Your example must pass `arthur` all of the arguments it expects (i.e. the value cannot be another function). ⟸

(c) Using the rules for the environment model defined in class, how many closures are created when we evaluate the expression `ford(AND(X,NOT Y))`? ⟸

5. (25 points)

Some of the following expressions can be made to evaluate to 42 by filling in $\boxed{\texttt{TYPE}}$ and $\boxed{\texttt{EXP}}$ correctly. For example, the expression

```
let val ford: TYPE = EXP in
    ford(21)
end
```

can be made to evaluate to 42 by having $\boxed{\texttt{TYPE}}$ = `int->int` and $\boxed{\texttt{EXP}}$ = `fn(x:int) => x*2`.

For each of the following expressions, specify $\boxed{\texttt{TYPE}}$ and $\boxed{\texttt{EXP}}$, or state that the expression cannot be made to evaluate to 42.

IMPORTANT NOTE: you may not use parameterized types or exceptions in your solutions, but you may use side effects. In addition, if $\boxed{\texttt{EXP}}$ is a function, the function must depend in a non-trivial way of all its inputs. This means, for example, that the function cannot be a constant.

(a)
```
let
  val ford: TYPE = EXP
  val x = ref (6*9)
  val y = ref x
in
  (ford(y);
   !x)
end
```
$\Longleftarrow$

(b)
```
let
  val ford: TYPE = EXP
  val x = ref [ref 6, ref 9]
  val y = ref x
in
  (ford(y);
   !(hd(!x)) * !(hd(tl(!x))))
end
```
$\Longleftarrow$

(c)
```
let
  val f: TYPE = EXP
in
  length ((f(f(f(f(f [()]))))) @ (f(f(f [()]))) @ (f [()]))
end
```
$\Longleftarrow$

(d)
```
let
  val ford: TYPE = EXP
  fun arthur n =
    let
      fun helper n x y =
      case n of
        0 => x
      | 1 => y
      | _ => helper (n-1) y (x+y)
    in
      helper n 0 1
    end (* end of arthur definition *)
in
  foldl ford 0 (map arthur [6, 9])
end
```
$\Longleftarrow$

(e)
```
let
 val ford: TYPE = EXP
in
 ford ! 42
end
```
Hint: recall that `!` is a unary operator, much like `not`. $\Longleftarrow$

6. (30 points)

For this problem we will use the usual definition of streams.

```
datatype 'a stream = Empty | Stream of 'a * (unit -> 'a stream)
```

Now consider the following function.

```
fun zaphod(s: int stream): int stream =
let
  val ford: int list ref = ref []
  fun arthur(): int =
  let
    val marvin =
       foldl (fn (v, m) => Int.min(v, m)) (hd (!ford)) (tl (!ford))
    val () = ford := List.filter (fn v => v <> marvin) (!ford)
  in
    marvin
  end (* end of arthur *)
  fun helper(s: int stream): int stream =
  case (length (!ford), s) of
    (0, Empty) => Empty
  | (_, Empty) => Stream(arthur(), fn() => helper Empty)
  | (n, Stream(v, tl)) => let
                            val marvin = if n = 100
                                           then arthur()
                                         else 0
                            val () = ford := v :: (!ford)
                          in
                            if n = 100
                              then Stream(marvin,
                                          fn() => helper (tl()))
                            else helper (tl())
                          end (* end of helper *)
in
  helper s
end
```

(a) What is the type of `marvin`? ⟸

(b) Give a short English description of what the function `arthur` does. ⟸

(c) Consider the following streams:

```
val s1 = Empty
val s2 = Stream(3, Stream(2, Stream(1, Empty)
```

What is the value of `zaphod(s1)`? What is the value of `zaphod(s2)`? ⟸

(d) Now consider the following expressions:

```
fun const(n) = Stream(n, fn() => const n)
fun const2(n,k) =
  if k = 0 then Empty
           else Stream(n, fn () => const2 n (k - 1))
val s3 = const 312
val s4 = const2 312 2003
```

What is the value of `zaphod(s3)`? What is the value of `zaphod(s4)`? ⟸

(e) Let S100 be the set of int stream values with the property that if s is in S100, then $s(i) < s(i + 100)$, for all $i \geq 0$, and that all values $s(i)$ are distinct. The notation $s(i)$ refers to the ith element of stream s; the first element of s, if it exists, is $s(0)$. For the purposes of this problem you should assume that the type int has no upper bound. (This is actually true in some programming languages, such as Scheme.) Give a short English description of what the function `zaphod` does when its input is an arbitrary stream in S100. $\Longleftarrow$