Name: ______________________________

ID number: ______________________________

CS 312, Fall 2002

Exam 1

October 17, 2002

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

There are four (4) problems on this exam. Please check now that you have a complete exam booklet with eight (8) numbered pages after 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 roughly by increasing difficulty.

This exam is closed book. No papers, books or notes may be used.

There is space provided to answer each question. You may use the back of the page, or request extra 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. (20 Points) For each of the following expressions determine what value would be returned by SML, or indicate that an error would occur.

   (a) $\Longleftarrow$
   ```
   let val x:int list = [4,2] in
       hd(x) + hd(tl(x))
   end
   ```

   (b) $\Longleftarrow$
   ```
   let val x:int list = [4,2] in
       hd(x) * tl(x)
   end
   ```

   (c) $\Longleftarrow$
   ```
   let val x:int = 4 in
       let val ford:int->int = fn(z:int) => x*z in
           let val x:int = 2 in
               ford(x)
           end
       end
   end
   ```

(d)
```
let val ford:int->int =
        fn(x:int) =>
           let val x:int = x + x in
             let val arthur:int->int = (fn(x:int) => x*4) in
               let val marvin:int->int = (fn(z:int) => x+z) in
                 arthur(marvin(x))
               end
             end
           end
    in
      ford(2)
end
```
⟸

(e)
```
let val ford:int->int =
  (fn(z:int) =>
    (fn(self:int->int) => z+4)(fn(self:int) => z+4))
   in
    let val marvin:(int->int)->int =
          (fn(arthur:int->int) => arthur(ford(2)))
      in
       marvin(fn(y:int) => y*3)
   end
end
```
⟸

2. (25 points)

   The procedure `mapsquares` takes a list of integers and returns a new list where each element of the new list is the square of the corresponding element in the input list. So, for example, we have

   `mapsquares([1,2,5,4])` $\Rightarrow$ `[1,4,25,16]`

   We will define `mapsquares([])` $\Rightarrow$ `[]`.

   (a) Give a short recursive definition of `mapsquares`. To simplify the next part of this problem, please do not use tail-recursion, pattern matching or case statements. $\Longleftarrow$

   (b) Prove that your definition of the `mapsquares` function is correct, using induction and the substitution model. You may assume that all ML arithmetic and list primitives perform correctly. In using the substitution model, you only need to show the expression you are evaluating, and any steps that involve either a substitution or the use of the variable you are doing induction on. To receive full credit you must justify each step. $\Longleftarrow$

3. (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 HHGTTG:[TYPE] = [EXP] in
    HHGTTG(21)
end
```

can be made to evaluate to 42 by
$\boxed{\texttt{TYPE}}$ = `int->int`
$\boxed{\texttt{EXP}}$ = `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 and justify your claim.

(a) (5 points)

```
let val HHGTTG:[TYPE] = [EXP] in
  let val ford:int list->int = (fn(x:int list) => hd(x)) in
    let val arthur:int list->int =  fn(y:int list) => hd(tl(y)) in
     (ford(HHGTTG)+1)*(arthur(HHGTTG)-1)
    end
  end
end
```

⟸

(b) (5 points)

```
let val HHGTTG:[TYPE] = [EXP] in
  let val ford:int list->int = (fn(x:int list) => hd(x)) in
    let val arthur:int list->int =  fn(y:int list) => hd(tl(y)) in
      ford(HHGTTG)*HHGTTG(arthur)
    end
  end
end
```

⟸

(c) (5 points)

```
let val HHGTTG: TYPE  =  EXP  in                                          ⟸
  let val ford:int list->int = (fn(x:int list) => hd(x)) in
    let val arthur:int list->int =  fn(y:int list) => hd(tl(y)) in
      (ford(HHGTTG))(arthur(HHGTTG))
    end
  end
end
```

(d) (10 points)

```
let val HHGTTG: TYPE  =  EXP  in                                          ⟸
  let val ford:int list->int = (fn(x:int list) => hd(x)) in
    let val arthur:int list->int =  fn(y:int list) => hd(tl(y)) in
      (HHGTTG(ford))(HHGTTG(arthur))
    end
  end
end
```

4. (30 points)

   The procedure `listsumsto` takes a (non-empty) list of integers and returns a new list where each element of the new list is the sum of all the previous elements of the input list. So, for example, we have

   `listsumsto([1,2,5,4])` $\Rightarrow$ `[1,3,8,12]`

   You may assume that `listsumsto` is only called on a non-empty list of integers. Hint: it may be useful to use `List.rev`, which reverses a list, and also to write one or more helper functions within the main functions you define.

   (a) Write a recursive (but not tail-recursive) definition of the function `listsumsto`. $\Longleftarrow$

(b) Write a tail-recursive definition of the function `listsumsto`. ⟸

(c) Recall that the definition of `foldl` is

```
fun foldl (f:'a*'b->'b) (accum:'b) (x:'a list) : 'b =
  case x of nil => accum
| hd::tl => foldl f (f(accum,hd)) tl
```

Write a non-recursive definition of the function `listsumsto` using a single call to `foldl`. $\Longleftarrow$