Name: _____

ID number: _____

CS 312, Fall 2003

Exam 1

October 16, 2003

Problem	1	2	3	4	Total
Grader					
Grade	/18	/24	/28	/30	

There are 4 problems on this exam. Please check now that you have a complete exam booklet with 9 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.

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. (18 Points) For each of the following expressions determine what value would be returned if you started up SML and typed this expression into the interpreter. Alternatively, indicate that an error would occur. (Note that you should assume each expression is typed into a brand-new SML interpreter.)

   (a)
   ```
   let                                                    ⟸
     val x = 42
     val y = [x,x]
   in
     hd(y)+x
   end
   ```

   Answer: 84

   (b)
   ```
   let                                                    ⟸
     val y = [4,2]
     val f = fn(x) => x-4
     val z = map f y
   in
     hd(y) * hd(tl(z))
   end
   ```

   Answer: 8

(c)
```
let
  val f = fn(x) => x-4
  val x = 2
in
  f(x)::[f(x)]
end
```
⟸

Answer: `[ 2, 2]`

(d)
```
let val f =
  fn(x) =>
    let val z = 2
    in
      fn(y) => (z*y)+x
    end
in
  f(4)(2)
end
```
⟸

Answer: 8

(e)
```
let
  val f = fn(x) => x-z
  val x = 2
  val z = 4
in
  f(x)::[f(x)]
end
```
⟸

Answer: Error (both x and z are unbound)

(f)
```
let
  val x = 42
  val f = fn(x) => x+42
  val x = 24
  val g = fn(x) => f(x)+x
in
  f(g(x))
end
```
⟸

Answer: 132

2. (24 Points) Some of the following expressions can be made to evaluate to `42` by filling in `TYPE` and `EXP` correctly. For example, the expression

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

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

For each of the following expressions, specify `TYPE` and `EXP`, or state that the expression cannot be made to evaluate to 42. Please note that for simplicity you may not use parameterized types or exceptions or exception handlers in your solutions.

(a)
```
let val HHGTTG:TYPE = EXP in
  foldl HHGTTG 1 [2,3,4]
end
```
⟸

`TYPE` = `(int*int)->int`
`EXP` = `(fn(x:int,y:int)=>if x = 4 then 42 else x*y)`

(b)
```
let val HHGTTG:TYPE = EXP in
  let val ford = (fn(x) => map hd x) in
     hd(ford(HHGTTG)) + hd(tl(ford(HHGTTG)))
  end
end
```
⟸

`TYPE` = `int list list`
`EXP` = `[[2,15],[40,14]]`

(c)
```
let
  val HHGTTG:TYPE = EXP
  val ford = fn(f,g) =>
       fn(x) => f(g(x+1)*20)
  val arthur = ford (hd(HHGTTG),hd(tl(HHGTTG)))
in
  arthur(4)
end
```

TYPE = `(int->int) list`
EXP = `[fn(x) => 42, fn(y) => y]`

(d)
```
let
  val HHGTTG:TYPE = EXP
  val ford = fn(query) =>
    if query(6*9) = 42 then 42 else raise Fail "not the answer!"
  val arthur = fn(question) => 42
  val marvin = fn(question) => 24
in
  HHGTTG(ford(arthur),ford(marvin))
end
```

Impossible, because functions evaluate their arguments. Full credit for stating that it is impossible because ford(marvin) will raise an exception.

3. (28 points) The procedure `countdown` takes two integers m, n and returns a list of length $m+1$ whose elements are $[m, m-1, m-2, \ldots, 1, n]$. So, for example,

```
countdown(3,4) ⇒ [3,2,1,4]
countdown(0,42) ⇒ [42]
```

(a) **(3 points)** Write a simple recursive definition of the function `countdown`. To simplify the next part of the problem, your definition should not be tail-recursive, and should not use pattern matching or `case` (use `if` and list primitives like `::` instead). ⟸

Sample solution:

```
fun countdown(m:int,n:int):int list =
  if (m = 0) then [n] else m::countdown(m-1,n)
```

(b) **(20 points)** Prove that your definition of the `countdown` function is correct, using induction and the substitution model. You may assume that all arithmetic and list primitives referenced in the definition of `countdown` perform correctly. To receive full credit you must justify each step. ⟸

1. Statement to be proved: `countdown`(m, n) `=` $[m, m-1, \ldots, 1, n]$. We could write this as P$[m]$, since the output recursively depends on m.

2. Induction variable: $m \in 0, 1, 2 \ldots$. Note that you can't do induction on n here!

3. Base case: We need to prove P[0], i.e.

`countdown`$(0, n)$ `=` $[n]$

By the RSM (substitution),

`countdown`$(0, n)$ `= if 0 = 0 then` $[n]$ `else 0::countdown(0-1,`n`)`

This is $[n]$ by the RSM (semantics of `=,if`), so P[0] is true.

4. Induction case: we need to prove P$[m]$ implies P$[m+1]$. In other words, we need to show:

`countdown`$(m+1, n)$ `=` $[m+1, m, m-1, \ldots, 1, n]$

assuming the Induction Hypothesis, which states that
`countdown`(m, n) `=` $[m, m-1, \ldots, 1, n]$

By the RSM (substitution), we have:

`countdown(`$m+1, n$`) = if` $m+1$ `= 0 then [`n`] else` $m+1$`::countdown(`$m+1$`-1,`n`)`

Since $m \in 0, 1, 2 \ldots$ we know that $m + 1 \neq 0$. Applying the RSM (semantics of `=,if`) we get:

`countdown(`$m + 1, n$`) =` $m + 1$`::countdown(`$m + 1$`-1,`n`)`

which of course yields

`countdown(`$m + 1, n$`) =` $m + 1$`::countdown(`m, n`)`

By the Induction Hypothesis this gives us

`countdown(`$m + 1, n$`) =` $m + 1$`::[`$m, m - 1, \ldots, 1, n$`]`

which by the RSM (semantics of `::`) gives

`countdown(`$m + 1, n$`) = [`$m + 1, m, m - 1, \ldots, 1, n$`]`

This is exactly what we wanted to prove.

(c) **(5 points)** Write a tail-recursive definition of the function `countdown`. ⟸

Sample solution:

```
fun countdowniter(m:int,n:int):int list =
  let fun iter(m0:int, ans:int list) =
    if (m0 = 0)
      then List.rev(n::ans)
    else iter(m0-1,m0::ans)
  in
    iter(m,[])
  end
```

4. (30 points) Now consider the problem of determining if a list of integers is the value of `countdown(`m, n`)` for two integers m, n. You will need to write a function `countdowninv(`l`)` that returns the m, n such that l `= countdown(`m, n`)`. If there are no such m, n the value of `countdowninv(`l`)` should be the value of the expression `nocount()` which we could define as follows:

```
fun nocount():(int*int) = raise Fail "No such countdown"
```

After your code, the SML interpreter should behave as follows:

```
-countdowninv([42]);
val it = (0,42) : int * int
-countdowninv(countdown(4,2));
val it = (4,2) : int * int
-countdowninv([42,42]);
uncaught exception Fail: No such countdown
```

You will write several versions of this function.

(a) **(10 points)** Write a non-recursive solution for `countdowninv`. (Hint: you are welcome to define and use helper functions.) ⟸

Sample solution:

```
fun countdowninv(lst:int list):(int*int) =
  let
    val (a,b) = (List.length(lst)-1, hd(List.rev(lst)))
  in
    if (null(lst) orelse countdown(a,b) <> lst) then nocount() else
(a,b)
  end
```

There is also a solution using fold.

(b) **(10 points)** Write a recursive, but not tail-recursive, definition of `countdowninv`.

Sample solution:

```
fun countdowninv(lst:int list):(int*int) =
  case lst of
    [] => nocount()
  | [last] => (0,last)
  | x::y => let val (a,b) = countdowninv(y) in
      if (x = a+1) then (x,b) else nocount()
    end
```

(c) **(10 points)** Write a tail-recursive definition of `countdowninv`. Hint: if the expressions E1 and E2 do not involve the function `iter`, then a function of the form

```
fun iter(lst:int list,x:int):(int*int) =
    if (x <> E1) then E2 else iter(tl(lst),x-1)
```

will be tail-recursive.

Sample solution:

```
fun countdowninv(lst:int list):(int*int) =
  case lst of
    [] => nocount()
  | [last] => (0,last)
  | start::y =>
      let fun iter(remain:int list,next:int):(int*int) =
        case remain of
          [last] => if (next <> 0) then nocount() else (start,last)
        | a::b => if (a <> next) then nocount() else iter(b,a-1)
      in
        iter(y,start-1)
    end
```