Name:____

CSE 341, Fall 2004, Midquarter Examination
1 November 2004

Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- **Please stop promptly at 1:20.**
- You can rip apart the pages, but please write your name on each page.
- There are a total of **60 points**, distributed unevenly among 5 questions (each with multiple parts).
- When writing code, style matters, but don't worry about indentation.

Advice:

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit.
- The questions are not necessarily in order of difficulty. **Skip around.**
- If you have questions, ask.
- Relax. You are here to learn.

Name:____

1. Consider this `datatype` for *non-empty* lists (but not built-in ML lists) of integers:

```
datatype t = One of int | More of int * t
```

   (a) **(4 points)** Write an ML function `length` that takes `t` and returns how many `int` values are in the `t`. Your solution must *not* be tail-recursive.

   Solution:

```
 fun length lst =
   case lst of
     One _ => 1
   | More(_,m) => 1 + length m
```

   (b) **(6 points)** Write an ML function `rev_map` that takes 3 arguments (as a tuple): (1) A function `f` from integers to integers, (2) a `t` called `acc`, and (3) a `t` called `lst`. The function should return a `t` that is the result of reversing `lst`, applying `f` to every `int` to the reversed list, and appending that result to `acc`. For example,

```
rev_map ((fn x => x+1), (More (0, One(1))), (More(3, More(4, One(5)))))
```

   should evaluate to:

```
More(6, More (5, More(4, More(0, One(1)))))
```

   Implement `rev_map` as a *tail-recursive* function that uses no helper functions.

   Solution:

```
fun rev_map (f,acc,lst) =
  case lst of
    One i => More(f i, acc)
  | More(i,tl) => rev_map(f,More(f i, acc),tl)
```

   (c) **(3 points)** What is the type of `rev_map`?

   Solution:

```
(int->int) * t * t -> t
```

Name:____

2. For each of the following programs, give the value that `ans` is bound to after evaluation. Underlining is just to help you see the differences between problems.

(a) **(3 points)**

```
fun f x =
  let val x = x + 1
      val y = x + 1
  in
    y + 1
  end
val x = 1
val ans = f x
```

Solution:
4

(b) **(3 points)**

```
fun f x =
  let val y = x + 1
      val x = x + 1
  in
    y + 1
  end
val x = 1
val ans = f x
```

Solution:
3

(c) **(3 points)**

```
fun f (x,y) =
  if x=10
  then (fn x => x + y)
  else (fn x => x - y)
val x = f(3,4)
val ans = x 10
```

Solution:
6

(d) **(4 points)**

```
fun f (x,y) =
    (fn x => if x=10
             then x + y
             else x - y)
val x = f(3,4)
val ans = x 10
```

Solution:
14

Name:____

3. Consider this ML function:

```
fun someFun (f,g,start,stop) =
  let fun loop n =
        (n <= stop) andalso ((f (g n)) orelse (loop (n + 1)))
  in
    loop start
  end
```

(a) **(5 points)** Fill in the blanks to give the type of `someFun`.
Hint: The solution has one type variable, which appears twice.

(____________________ *

____________________ *

____________________ *

____________________) -> ____________________

Solution:
`('a->bool)*(int->'a)*int*int->bool`

(b) **(7 points)** What does `someFun` compute? (Describe what it computes from a caller's perspective, not how `someFun` works. Start your answer with "`someFun(f,g,start,stop)` evaluates to ______ if and only if . . . ".)

Solution:
`someFun(f,g,start,stop)` evaluates to `true` if and only if there exists a number n between `start` and `stop` (inclusive) such that applying `f` composed with `g` to n evaluates to `true`.

(c) **(2 points)** Fill in the blank so evaluating this programs produces `true`:

```
val x = _______
someFun((fn z => z = 6), (fn y => x * y), x, (x + 1))
```

Solution:
2

Name:____

4. Each pair of expressions below is *not* totally contextually equivalent. Briefly explain why. (Underlining just emphasizes differences.)

   (a) **(3 points)**
   `let val x = 0 in x end` and `let val x = (f 3) - (f 3) in x end`

   (b) **(4 points)**
   `(fn x:int => fn z:int => x - y) y` and `(fn x:int => fn y:int => x - y) y`

Solution:

(a) Evaluating `(f 3)` might have an effect (infinite-loop, exception, I/O, etc.)

(b) The first expression evaluates to a function that always returns 0. The second expression evaluates to a function that subtracts its argument from `y`. For example, in an environment where `y` maps to 3, the first expression is equivalent to `(fn z => 0)` and the second expression is equivalent to `(fn y => 3 - y)`.

Name:____

5. Consider this ML structure definition:

```
structure M :> MSIG =
struct
  type one_or_two = bool * int * int
  fun abs_val i = if i < 0 then ~i else i
  fun mkOne i = (false,(abs_val i),~1)
  fun mkTwo (i,j) = (true,(abs_val i),(abs_val j))
  fun last (x : one_or_two) = if #1 x then #3 x else #2 x
end
```

(a) **(4 points)** Why is the definition of type `one_or_two` bad style? Suggest a different way to program this idea that uses ML's features more appropriately. (Hint: We are asking about the type definition. The fact that `last` doesn't use pattern-matching is *not* the answer.)

(b) For each of the following `MSIG` definitions, determine if a client can make a call to `last` evaluate to a negative number. **Explain briefly.**

i. **(3 points)**

```
signature MSIG =
sig
  type one_or_two = bool * int * int
  val mkOne : int -> one_or_two
  val mkTwo : int * int -> one_or_two
end
```

ii. **(3 points)**

```
signature MSIG =
sig
  type one_or_two = bool * int * int
  val mkTwo : int * int -> one_or_two
  val last  : one_or_two -> int
end
```

iii. **(3 points)**

```
signature MSIG =
sig
  type one_or_two;
  val mkOne : int -> one_or_two
  val mkTwo : int * int -> one_or_two
  val last  : one_or_two -> int
end
```

Solution:

(a) Because `one_or_two` is a "one of" type, so a `datatype` binding, e.g., `datatype one_or_two = One of int | Two of int * int` is better style.

(b) i. No, `last` is not in the signature; no client can cause `last` to be called with any arguments.

ii. Yes, a client can just do `M.last(true,0,~1)`.

iii. No, `one_or_two` is abstract, so any value of this type was produced by `mkOne` or `mkTwo`. Looking at these functions, that means either the `bool` is `false` and the second `int` is non-negative, or the third `int` is non-negative. In either case, `last` evaluates to a non-negative number.