

Name: _____

CSE341, Spring 2013, Midterm Examination
May 3, 2013

Please do not turn the page until 12:30.

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 staple them back together before you leave.
- There are **100 points** total, distributed **unevenly** among **6** questions (all with multiple parts).
- When writing code, style matters, but don't worry much 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.** Make sure you get to all the problems.
- If you have questions, ask.
- Relax. You are here to learn.

Name: _____

1. This problem uses this datatype binding for *ternary trees*, where a ternary tree is a tree where all non-leaves have exactly three children:

```
datatype int_ternary_tree = Leaf of int
                           | Node of int
                               * int_ternary_tree
                               * int_ternary_tree
                               * int_ternary_tree
```

- (a) (8 points) Write an ML function `to_list` of type `int_ternary_tree -> int list`. The result should have every number that appears anywhere in the argument (and no other numbers). If a number appears n times in the argument, then it also appears n times in the result. The order of numbers in the result does not matter.
Use no helper functions other than `::` and `@`.
- (b) (10 points) Write a second version of `to_list` that:
- Does *not* use `@` (and not your own reimplementations of it)
 - *Does* use a locally-defined helper function of type `int_ternary_tree * int list -> int list`
 - Does *not* need to produce a list in the same order as your answer in part (a).
- (c) (3 points) Is your answer to part (a) tail-recursive? Explain in 1-2 sentences.
- (d) (3 points) Is your answer to part (b) tail-recursive? Explain in 1-2 sentences.

Solution:

```
(a) fun to_list t =
      case t of
        Leaf i => [i]
      | Node(i,a,b,c) => i :: ((to_list_a a) @ (to_list_a b) @ (to_list_a c))
```

```
(b) fun to_list t =
      let
        fun f (t,acc) =
          case t of
            Leaf i => i::acc
          | Node(i,a,b,c) => f(a,f(b,f(c,i::acc)))
      in
        f(t,[])
      end
```

- (c) No, after the recursive calls, the caller passes the result to `@` rather than immediately returning the result.
- (d) No, one of the 3 recursive calls to `f` is a tail call, but the other two are not because the results are passed to other calls before the caller returns.

Name: _____

2. This problem uses this ML code:

```
exception Foo

fun f1 (xs,ys) =
  case (xs,ys) of
    (x::[], _) => x
  | (_, z::[]) => z
  | (x::y::_, _) => y
  | _ => raise Foo

fun f2 (xs,ys) =
  case (xs,ys) of
    (x::[], _) => x
  | (x::y::_, _) => y
  | (_, z::[]) => z
  | _ => raise Foo

fun f3 (xs,ys) =
  case (xs,ys) of
    (x::y::_, _) => y
  | (_, z::[]) => z
  | (x::[], _) => x
  | _ => raise Foo
```

- (a) (5 points) Give an **a** and **b** such that **a** and **b** are lists with no numbers duplicated (not even across the two lists) and **f1(a,b)**, **f2(a,b)**, and **f3(a,b)** all evaluate to 341.
- (b) (4 points) Give an **a** and **b** such that **a** and **b** are lists with no numbers duplicated (not even across the two lists) and **f1(a,b)** and **f2(a,b)** evaluate to 341 but **f3(a,b)** does not.
- (c) (4 points) Give an **a** and **b** such that **a** and **b** are lists with no numbers duplicated (not even across the two lists) and **f2(a,b)** and **f3(a,b)** evaluate to 341 but **f1(a,b)** does not.

Solution:

- (a) Three approaches:
- **a** is [] or a list with three or more elements (with no 341 and no duplicates) and **b** is [341]
 - **a** is [341] and **b** does not have 1 element (and no duplicates or 341)
 - **a** has two or more elements with 341 second and **b** does not have 1 element (with no duplicates in the lists) has 341 either first or second and **b** does not have 1 element (with no duplicates in the lists)
- (b) **a** is [341] and **b** is any one-element list not containing 341
- (c) **a** is any two-element list with 341 second— and **b** is any one-element list (with no duplicates in the lists)

Name: _____

3. For each of the following programs, give the value **ans** is bound to after evaluation.

(a) (5 points)

```
fun f x y z = if z > 0 then (fn w => w + x + y) else (fn w => w + x - y)
val a = 1
val b = 2
val c = f b a
val d = c ~7
val ans = d 4
```

(b) (5 points)

```
fun f p =
  let
    val x = 3
    val y = 4
    val (z,w) = p
  in
    (z (w y)) + x
  end
val x = 1
val y = 2
val ans = f((fn z => x + z), (fn x => x + x))
```

(c) (5 points)

```
fun f x = x + 7

fun g y =
  if y > 0
  then (f (y-1)) + 1
  else 4
and f y = (* notice the keyword and on this line *)
  if y > 0
  then (g (y-1)) + 2
  else 5

val ans = f 3
```

Solution:

(a) 5

(b) 12

(c) 9

Name: _____

4. (14 points) This problem uses this ML code:

```
datatype my_int_list = Empty
                    | Cons of int * my_int_list

fun foo g a x =
  case x of
    Empty => a
  | Cons(i,x') => foo g (g(a,i)) x'
```

- (a) By using `foo` but not using any fun-bindings (you can use val-bindings and anonymous functions), bind to `first_odd` a function of type `my_int_list -> int` that returns the odd number closest to the beginning (head) of the `my_int_list`, or 0 if the `my_int_list` contains no odd numbers.
- (b) By using `foo` but not using any fun-bindings (you can use val-bindings and anonymous functions), bind to `last_odd` a function of type `my_int_list -> int` that returns the odd number closest to the end of the `my_int_list`, or 0 if the `my_int_list` contains no odd numbers.

If the no-fun-bindings requirement is confusing you, use a fun-binding for some partial credit, but still use `foo` as a helper function.

Solution:

- (a)

```
val first_odd = foo (fn (a,i) => if a=0 andalso i mod 2 = 1
                                then i
                                else a)
                                0
```
- (b)

```
val last_odd = foo (fn (a,i) => if i mod 2 = 1
                                then i
                                else a)
                                0
```

Name: _____

5. (a) (11 points) Without using any helper functions, write an ML function `filter_increasing`, which works as follows:
- It takes three arguments *in curried form*: (1) a function `f` that takes list elements and returns integers, (2) an integer `i`, and (3) a list `xs`.
 - It returns a list that contains a subset of the elements in `xs` in the same order they appear in `xs`.
 - An element of `xs` is in the output if and only if `f` applied to the element produces a number greater than `i` and greater than the number produced by `f` for all elements earlier (closer to the head) in the list.
- (b) (5 points) What is the type of `filter_increasing`?

Solution:

- (a)

```
fun filter_increasing f i xs =
  case xs of
    [] => []
  | x::xs' =>
    let
      val j = f x
    in
      if j > i
      then x :: filter_increasing f j xs'
      else filter_increasing f i xs'
    end
```
- (b) `('a -> int) -> int -> 'a list -> 'a list`

Name: _____

6. (18 points) This problem uses this ML signature definition:

```
signature S =  
sig  
  type t  
  (* one more line here as described below *)  
end
```

The comment in the definition above can be replaced by any *one* of the following:

```
(* 1 *) val f : int * int -> bool  
(* 2 *) val f : int -> int -> bool  
(* 3 *) val f : int * 'a -> bool  
(* 4 *) val f : t * t -> bool  
(* 5 *) val f : t * int -> bool  
(* 6 *) val f : t * 'a -> bool
```

Now suppose we have a structure definition like this:

```
structure M :> S =  
struct  
  type t = int  
  fun f ...  
end
```

For each different definition of `f` below, list exactly which types for `f` listed above would cause the signature to match, meaning `M` would type-check with signature `S`. For example, an answer could be, “1, 3, and 4” where the numbers refer to the numbers in comments above.

- (a) `fun f (x,y) = x > y andalso y > 3`
- (b) `fun f (x,y) = x > 7`
- (c) `fun f (x,y) = y > 7`
- (d) `fun f (x,y) = if x > y then 34 else 42`
- (e) `fun f x = x > 7`
- (f) `fun f x = true`

Solution:

- (a) 1, 4, 5
- (b) 1, 3, 4, 5, 6
- (c) 1, 4, 5
- (d) none
- (e) none
- (f) 1, 3, 4, 5, 6