CSE 341 : Programming Languages
Midterm, Spring 2014

Please do not turn the page until 12:30.

Rules:

- Closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- Please stop promptly at 1:20.
- You can separate pages, but please staple them back together before you leave.
- There are 100 points total, distributed unevenly among 6 questions.
- When writing code, style matters, but don't worry too 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 in order of difficulty. Skip around. Get to all the problems.
- If you have questions, ask.
- Don't worry too much; you're here to learn. You are smart and can totally do this!!!
1. (25 points) In this question we will use \texttt{map} and \texttt{fold} over lists to implement \texttt{tmap} and \texttt{tfold} over variable arity trees. Assume these implementations of \texttt{map} and \texttt{fold} for lists:

```verbatim
(* map : (\texttt{a -> b}) -> \texttt{a list -> b list} *)
fun map f [] = []
  | map f (h::t) = f h :: map f t

(* fold : (\texttt{a -> b -> b}) -> \texttt{b -> a list -> b} *)
fun fold f base [] = base
  | fold f base (h::t) = f h (fold f base t)
```

Consider this implementation of variable arity trees:

```verbatim
datatype \texttt{a tree} = Node of \texttt{a * (a tree list)}
```

How should we fill in the blank to map function \texttt{f} over an entire tree?

```verbatim
(* tmap : (\texttt{a -> b}) -> \texttt{a tree -> b tree} *)
fun tmap f (Node (x, ts)) = __________________________
```

(9 points) Circle the correct way to fill in the blank (only one of the options is correct):

(a) map f (x :: tmap f ts)

(b) Node (f x, map f (tmap ts))

(c) map f (tmap f (x :: ts))

(d) Node (f x, tmap (map f) ts)

(e) Node (f x, map (tmap f) ts)

(f) map f (x :: (tmap map) f ts)
How should we fill in the blank to fold function $f$ with $\text{base}$ over an entire tree?

\[
(* \quad \text{tfold} : (\text{a} \rightarrow \text{b} \rightarrow \text{b}) \rightarrow \text{b} \rightarrow \text{a tree} \rightarrow \text{b} *)
\]
\[
\text{fun tfold f base (Node (x, ts)) = ________________}
\]

(9 points) Circle the correct way to fill in the blank (only one of the options is correct):

(a) $f$ $x$ (tfold (fold (fn $t$ => fn acc => $f$ $t$ acc) base $t$s))

(b) $f$ $x$ (fold (fn acc => fn $t$ => tfold $f$ acc $t$) $t$s base)

(c) Node ($x$, tfold $f$ (fold (tfold $f$) $t$s))

(d) $f$ $x$ (fold (fn $t$ => fn acc => tfold $f$ acc $t$) base $t$s)

(e) $f$ $x$ (tfold (fn $t$ => fn acc => fold $f$ $t$ acc) base $t$s)

(f) Node ($f$ $x$, fold (tfold $f$) base $t$s)

(4 points) Use $\text{foldt}$ and the $\text{add}$ function below to fill in the blank for $\text{sumt}$, a function which adds up all the ints in an int tree. Note that $\text{sumt}$ uses a $\text{val}$ binding!

fun add $a$ $b$ = $a$ + $b$

\[
\text{val sumt = ________________}
\]

(3 points) Fill in the blank to show the type of $\text{sumt}$:

\[
\text{sumt : ________________}
\]
2. **(15 points)** Rewrite this function to be tail recursive (keep the same order!):

```haskell
fun pairUp x [] = []
    | pairUp x (h::t) = (x, h) :: pairUp x t
```

Rewrite this function to be tail recursive (keep the same order!):

```haskell
fun xprod [] ys = []
    | xprod (x::xs) ys = pairUp x ys @ xprod xs ys
```
3. (15 points) Consider the following datatype:

```haskell
datatype SkipList = Null | Node of int * SkipList * SkipList
```

We can use SkipList to represent lists where we can “skip ahead” to later parts of a list. For example the bindings below represent the following list:

```
val n6 = Node(6, Null, Null)
val n5 = Node(5, n6, Null)
val n4 = Node(4, n5, n6)
val n3 = Node(3, n4, Null)
val n2 = Node(2, n3, Null)
val n1 = Node(1, n2, n3)
```

Consider these two functions which attempt to flatten a SkipList into an ordinary list:

```haskell
fun flatten slist =
  case slist of
    Node (x, s1l, s12) => x :: (flatten s12)
  | Node (x, s1l, Null) => x :: (flatten s1l)
  | Null => []

exception NullListError

fun flatten_again slist =
  case slist of
    Node(x, s1l, Node(a, b, c)) => x :: (flatten_again s1l)
  | Node(x, s1l, Null) => x :: (flatten_again s1l)
  | Node(x, Null, Null) => [0]
  | Null => raise NullListError
```
(6 points) Assuming the implementation of `fold` and `add` from Problem #1 and the definitions above, what value will `sum1` be bound to below? If `sum1` will fail to evaluate due to an uncaught exception, write the name of the thrown exception in the blank.

``` scala
val sum1 = fold add 0 (flatten n1)
```

`sum1` = ______________________________

What value will `sum2` be bound to below? If `sum1` will fail to evaluate due to an uncaught exception, write the name of the thrown exception in the blank.

``` scala
val sum2 = fold add 0 (flatten_again n1)
```

`sum2` = ______________________________

(5 points) Provide a SkipList built with the `Node` constructor that will cause `flatten_again` to throw an exception, or if that is not possible explain why.

(4 points) Using all the same lines in `flatten`, but in a different order, write a function `flatten_yet_again` such that `flatten_yet_again n1` evaluates to `[1, 3, 4, 6]`:

``` scala
fun flatten_yet_again slist =

____________________________________________

____________________________________________

____________________________________________

____________________________________________
```

``` scala
```
4. (15 points) This question has three parts. We treat each part as though it were in its own separate namespace: bindings defined in previous parts are not valid in subsequent parts.

(5 points) Assuming the implementation of `map` from Problem #1, what is `ans` bound to after this code executes?

```rng
val (a, b) = (2, 4)
val add1 = (fn x => x + 1)
val times5 = (fn x => x * 5)
val square = (fn x => x * x)

fun f x y z =
  let
    val g = (fn (a, b') => a b)
  in
    y g x
  end

val foo = [(add1, true), (times5, false), (square, true)]
val ans = f foo map (fn x => [x, x])
```

ans = ______________________________________________

(5 points) Consider the following bindings. What will `ans` be bound to after this code executes?

```rng
val (a, b, c, x, y) = (2, 4, 6, 8, 10)
fun f x y =
  (let
    val x = y
    val b = a
    val b = b
  in
    c * b - b
  end) + x + b

val ans = y + f 3 5 - x
```

ans = ______________________________________________
(5 points) Consider the following two bindings:

```ml
fun h f = fn x => f x * f x
val v = h (h (h (fn y => y * y)))
```

Is `v` an int or a function? If it is an int, write its value. If it is a function, write its type and describe what the function computes.

(Optional bonus problem: 3 extra credit points) The bindings below define an int called `num`, and four functions called `f`, `g`, `h`, and `factorial`, where `factorial` is the familiar factorial function. Using each of `f`, `g`, `h`, `factorial`, and `num` exactly once, write an expression in the blank that will make it so that `ans` is bound to the factorial of `num`.

```ml
val num = 5

fun f a b c d = b a d c
fun g a b c = c a b
fun h a b = b a
fun factorial 0 = 1
    | factorial n = n * factorial (n - 1)

ans = _____________________________________________
```
5. (15 points) Consider this program:

```ml
val x = ref 0

fun foo y = 
  let 
    val _ = x := (!x + 1);
    val _ = print (Int.toString (!x) ^ " ")
  in
    !x + y
  end

val _ = print (Int.toString (foo 1) ^ " ")
val _ = print (Int.toString (foo 1) ^ " ")
val _ = x := 5
val _ = print (Int.toString (foo 1) ^ " ")
val x = ref 10
val _ = print (Int.toString (foo 1) ^ " ")
```

(8 points) What will it print? (Only one option is correct.)

(a) 0 1 1 2 5 6 6 7
(b) 1 2 1 2 1 2 1 2
(c) 1 2 2 3 6 7 11 12
(d) 1 2 1 2 6 7 6 7
(e) 1 2 2 3 6 7 7 8
(f) 1 2 2 3 3 4 11 12
Now consider this program:

```ocaml
fun bar y = 
  let
    val z = ref 0
    val _ = z := !z + 1
    val _ = print (Int.toString (!z) ^ " ")
  in
    !z + y
  end

val _ = print (Int.toString (bar 1) ^ " ")
val _ = print (Int.toString (bar 1) ^ " ")
val z = ref 10
val _ = print (Int.toString (bar 1) ^ " ")
```

(7 points) What will it print? (Only one option is correct.)

(a) 0 1 1 2 2 3
(b) 1 2 1 2 1 2
(c) 1 2 2 3 11 12
(d) 1 2 1 2 11 12
(e) 1 2 2 3 3 4
6. **(15 points)** Implement a module satisfying this signature:

```ml
signature STACK = sig
  type 'a t
  exception Empty
  val empty : 'a t
  val push : 'a -> 'a t -> 'a t
  val pop : 'a t -> 'a * 'a t
end
```

Your implementation should satisfy the following two properties:

1. `pop empty` should raise the `Empty` exception
2. `pop (push x stack)` should return `(x, stack)`

*(Hint: use lists!)*