Name:____

CSE341 Autumn 2018, Midterm Examination
October 29, 2018

Please do not turn the page until 12:30.

Rules:

- The exam is closed-book, closed-note, etc. except for *one* side of one 8.5x11in piece of paper.
- **Please stop promptly at 1:20.**
- There are **100 points**, distributed **unevenly** among **6** questions (all with multiple parts):
- **The exam is printed double-sided.**

Advice:

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit. But clearly indicate what is your final answer.
- The questions are not necessarily in order of difficulty. Skip around. Make sure you get to all the questions.
- If you have questions, ask.
- Relax. You are here to learn.

Name:____

1. (**23** points) This problem uses this datatype binding, where a `maze` involves any number of "choices" on the way "forward" with each "path" ending either successfully (`Finish`) or not (`DeadEnd`). A maze can have more than one `Finish`. "Solving" a maze means following any path that leads to `Finish`.

```
datatype maze =
           Finish
         | DeadEnd
         | Forward of maze * maze (* a pair of ("if going left", "if going right") *)
```

(a) Write a function `has_a_solution` of type `maze -> bool` that evaluates to true if and only if there is at least one path that ends with `Finish`.

(b) Given the additional datatype binding `datatype dir = Left | Right`, write a function `solve_maze` of type `maze -> dir list option`. As an example, `solve_maze (Forward(DeadEnd,Forward(Finish,Forward(DeadEnd,DeadEnd)))` would evaluate to `SOME [Right,Left]` because the maze can be "solved" by going right at the first `Forward` and left at the second `Forward`. Return `NONE` only if the maze has no solution. If a maze has multiple solutions, evaluate to `SOME xs` where `xs` indicates any one way to solve the maze. Hint: You need 2 or 3 case expressions, preferably 3.

(c) Give a value `v` such that the only correct result for `solve_maze v` is `SOME [Left,Left]`.

(d) Consider this alternate datatype for representing a maze:

```
datatype maze2 =
          End of bool (* true means finish; false means dead-end *)
        | Branch of maze2 list (* any number of next paths *)
```

Write a function `maze_to_maze2` of type `maze -> maze2` that produces a `maze2` that represents the same choices as the `maze` argument. Hint: All lists in the result will have length 2.

The next page is blank in case you need more room.

Solution:

(a)
```
fun has_a_solution m =
    case m of
       Finish => true
     | DeadEnd => false
     | Forward(m1,m2) => has_a_solution m1 orelse has_a_solution m2
```

(b)
```
fun solve_maze m =
    case m of
        Finish => SOME []
      | DeadEnd => NONE
      | Forward(m1,m2) => (* also fine to investigate right first or both *)
        case solve_maze m1 of
            NONE => (case solve_maze m2 of
                         NONE => NONE
                       | SOME xs => SOME (Right :: xs))
          | SOME xs =>  SOME (Left :: xs)
```

(c) `Forward(Forward(Finish,DeadEnd),DeadEnd)`

(d)
```
fun maze_to_maze2 m =
    case m of
        Finish => End true
      | DeadEnd => End false
      | Forward(m1,m2) => Branch [maze_to_maze2 m1, maze_to_maze2 m2]
```

Name:____

More room if needed for Problem 1.

Name:____

2. (**18** points) This problem considers the problem of writing a function of type `'a list * 'a list -> bool` that evaluates to `true` if and only if the first argument is longer-or-the-same-length-as the second argument. Here is a correct implementation:

```
fun longer (xs,ys) =
  case (xs,ys) of
     (_,[]) => true                  (* line 1 *)
   | ([],_) => false                 (* line 2 *)
   | (_::xs,_::ys) => longer(xs,ys)  (* line 3 *)
```

(a) For of the following alternate orders of the branches, indicate one of the following:
(A) The function would still be correct.
(B) The function would still type-check (no unreachable branch) but would no longer be correct.
(C) The function would no longer type-check (due to an unreachable branch).

i. line 2; then line 1; then line 3

ii. line 3; then line 1; then line 2

iii. line 3; then line 2; then line 1

iv. line 1; then line 3; then line 2

(b) Now consider the original order again but consider adding a fourth branch `| ([],[]) => true`. For each of the following positions for this extra branch, indicate (A), (B), or (C) as in the previous problem:

i. before line 1 (ignore the syntax issue that the first branch has no `|` character and line 1 would need one)

ii. between lines 1 and 2

iii. between lines 2 and 3

iv. after line 3

(c) Reimplement `longer` with a one-line `fun` binding using `List.length`.

Solution:

(a) i. B
ii. A
iii. B
iv. A

(b) i. A
ii. C
iii. C
iv. C

(c) `fun longer(xs,ys) = List.length xs >= List.length ys`

Name:____

3. (**13** points) For each of the following programs, if the program does not type-check answer "NO", else indicate what `ans` would be bound to after the program runs. Each part (a)-(d) is a separate program but (b), (c), and (d) all use this datatype binding:

```
datatype foo = A of int | B of string * foo
```

(a)
```
val y = 17
fun f x =
  let
      val z = y
  in
      (fn q => z + q + x)
  end

val y = 3
val ans = (f 8) y
```

(b)
```
fun g (x,b) =
  if b
  then A x
  else B (x, A 0)

val ans = g (3,true)
```

(c)
```
exception UhOh
fun m x =
  case x of
      A i => if i=7 then raise UhOh else 34
    | B(_,r) => m r

val ans = m (B("hi",B("bye", A 6))) handle UhOh => 19
```

(d)
```
val x = 3
fun h f = f x
val ans = h (A o (fn x => x+2)) (* recall o is function composition *)
```

Solution:

(a) `28`

(b) NO

(c) `34`

(d) `A 5`

Name:____

4. (**18** points)

   (a) Write a function `map_index` of type `(int * 'a -> 'b) -> 'a list -> 'b list` (notice the arguments are curried but the first argument takes a pair). `map_index` behaves like `map` except when the first argument is passed the i^{th} element of the second argument, it is also passed i (starting with 1 for the first element of the list). Use one locally-defined helper function and no other helper functions.

   (b) Use a `val` binding and a partial application of `map_index` to define `numbered`, a function of type `string list -> string list` that puts each string's position, a colon, and a space at the beginning of it. For example, `numbered ["hi", "bye", "Dan"]` would evaluate to `["1: hi", "2: bye", "3: Dan"]`. Hints: `Int.toString`, `^`.

   (c) Use a `val` binding and a partial application of `map_index` to define `redact_evens`, a function of type `string list -> string list` where the strings at odd-numbered list positions are in the output list unchanged and the strings at even-numbered list positions are replaced in the output list by the empty string `""`.

Solution:

```
fun map_index f xs =
  let
      fun aux i xs = (* can be curried or tupled *)
        case xs of
            [] => []
          | x::xs => (f (i,x))::(aux (i+1) xs)
  in
      aux 1 xs
  end

val numbered = map_index (fn (i,x) => Int.toString i ^ ": " ^ x)

val redact_evens = map_index (fn (i,x) => if i mod 2 = 0 then "" else x)
```

Name:____

5. (**8** points) Recall:

- `List.foldl` has type `('a * 'b -> 'b) -> 'b -> 'a list -> 'b`
- `List.filter` has type `('a -> bool) -> 'a list -> 'a list`

(a) Complete this function definition (by replacing the ... with some number of expressions) so that `rev_filter` is like `List.filter` except the result list is in the reverse order. Do not use `List.filter`.

```
fun rev_filter f = List.foldl ...
```

(b) In at most one sentence, give a reason your `rev_filter` is likely to be faster than `List.filter`.

Solution:

(a) `fun rev_filter f = List.foldl (fn (x,acc) => if f x then x::acc else acc) []`

(b) `List.foldl` is tail-recursive but `List.filter` is not.

Name:____

6. (**20** points) This problem considers an ML module `MinMaxList` and a signature `MINMAXLIST`. They are on the next page. Separate that page from your exam and do *not* turn it in.

   (a) Answer these questions about the `min` and `max` functions defined in `MinMaxList`.

      i. What is the type of `min` inside the module?

      ii. What is the type of `min` outside the module?

      iii. What is the type of `max` inside the module?

      iv. What is the type of `max` outside the module?

      v. Which is faster, `min` or `max`?

   (b) Given the signature `MINMAXLIST`:

      i. Can a client cause `min` or `max` to raise an exception?

      ii. Can a client cause `max` to return a number that isn't the maximum number in its argument?

      iii. Can a client cause `min` to return a number that isn't the minimum number in its argument?

   (c) Repeat part (b) but assuming we *replace* the line `type my_int_list` with
   `type my_int_list = int list`

      i. Can a client cause `min` or `max` to raise an exception?

      ii. Can a client cause `max` to return a number that isn't the maximum number in its argument?

      iii. Can a client cause `min` to return a number that isn't the minimum number in its argument?

   (d) Repeat part (b) but assuming we start with the original `MINMAXLIST` (not any changes from previous parts) and *add* this line to the signature:
   `val empty : my_int_list`

      i. Can a client cause `min` or `max` to raise an exception?

      ii. Can a client cause `max` to return a number that isn't the maximum number in its argument?

      iii. Can a client cause `min` to return a number that isn't the minimum number in its argument?

   (e) Repeat part (b) but assuming we start with the original `MINMAXLIST` (not any changes from previous parts) and *add* this line to the signature:
   `val cons : int * my_int_list -> my_int_list`

      i. Can a client cause `min` or `max` to raise an exception?

      ii. Can a client cause `max` to return a number that isn't the maximum number in its argument?

      iii. Can a client cause `min` to return a number that isn't the minimum number in its argument?

Solution:
See next page.

Solution:

(a)
   i. `int list -> int`
   ii. `MinMaxList.my_int_list -> int`
   iii. `'a list -> 'a`
   iv. `MinMaxList.my_int_list -> int`
   v. `max`

(b)
   i. no
   ii. no
   iii. no

(c)
   i. yes
   ii. yes
   iii. no

(d)
   i. yes
   ii. no
   iii. no

(e)
   i. no
   ii. yes
   iii. no

These definitions are used in Problem 6. Rip this page out from the rest of the exam. Do not put answers on this page and do not turn it in.

```
signature MINMAXLIST =
sig
type my_int_list
val new : int -> my_int_list
val add : int * my_int_list -> my_int_list
val max : my_int_list -> int
val min : my_int_list -> int
end

structure MinMaxList :> MINMAXLIST =
struct

type my_int_list = int list

exception Bad

val empty = []

fun cons (i,xs) = i::xs

fun new i = i::[]

fun add (i,xs) =
  case xs of
     [] => i::[]
   | j::ys => if i < j then j::i::ys else i::xs

fun min xs =
  case xs of
      [] => raise Bad
    | i::[] => i
    | i::ys => let val m = min ys in if i < m then i else m end

fun max xs =
  case xs of
      [] => raise Bad
   | i::_ => i

end
```