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 5 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.
1. This problem uses this datatype binding, where a value of type `points` describes a set of points on the plane, i.e., a 2-D plot with an x-axis and a y-axis.

```
datatype points = Point of real * real
| Seg of real * real * real * real
| Union of points * points
| Shift of points * real * real
```

- `Point(x,y)` represents the point \((x,y)\).
- `Seg(x1,y1,x2,y2)` represents all points on the line segment with endpoints \((x1,y1)\) and \((x2,y2)\).
- `Union(s1,s2)` represents all points represented by \(s1\) unioned with all points represented by \(s2\).
- `Shift(s,dx,dy)` represents the points represented by \(s\) after shifting them to the right by \(dx\) and up by \(dy\).

Note: we did not use type `real` much in class, but you can use arithmetic operations (e.g., `+`) and comparison operations (e.g., `>` as expected.

(a) (12 points) Write an ML function `rightmost` of type `points -> real * real` such that `rightmost s` returns the point in the set represented by \(s\) with the largest \(x\)-coordinate. (You can resolve ties however you wish.) Notice the result type is `real * real`, the \(x\)-coordinate and \(y\)-coordinate.

(b) (12 points) Write an ML function `max_shifts` of type `points -> int` that given \(s\) computes the maximum number of shifts that apply to a single “point” or “segment” in \(s\). Note this is not necessarily the number of `Shift` constructors in \(s\). For example, the correct answer for

```
Union(Shift(Point(0.0,0.0),1.0,1.0),
     Shift(Union(Shift(Point(2.0,2.0),1.0,1.0),
             Shift(Shift(Seg(3.0,4.0,5.0,6.0),7.0,8.0),9.0,10.0)),
        20.0,75.0))
```

is 3 because the one segment is under three `Shift` constructors, including the one outside the nested `Union`.

Solution:
See next page
Solution:

(a) fun rightmost s =
    case s of
    | Point p => p
    | Seg(x1,y1,x2,y2) => if x1 > x2 then (x1,y1) else (x2,y2)
    | Union(s1,s2) =>
      let
      in
      end
    | Shift(s1,dx,dy) =>
      let
      in
      end
(b) fun max_shifts s =
    case s of
    | Point _ => 0
    | Seg _ => 0
    | Union(s1,s2) => Int.max(max_shifts s1, max_shifts s2)
    | Shift(s,_,_) => 1 + max_shifts s

You can also implement the Union case without using the standard library with:

let
  val i1 = max_shifts s1
  val i2 = max_shifts s2
in
  if i1 > i2 then i1 else i2
end
2. This problem uses these two similar but different functions:

```haskell
fun f1 (xs,ys) =  
    case (xs,ys) of  
    ([], []) => []  
    | (x::xs', y::ys') => (x,y)::(f1(xs',ys'))  
    | (x::xs', []) => []  
    | ([], y::ys') => []  

fun f2 (xs,ys) =  
    case (xs,ys) of  
    ([],[]) => []  
    | (x::xs', y::ys') => (x,y)::(f2(xs',ys'))  
    | (x::xs', []) => (x,0)::(f2(xs',[]))  
    | ([], y::ys') => (0,y)::(f2([],ys'))  
```

(a) (5 points) Fill in the blanks so that c1 and d1 are both bound to \([(2,2),(1,1),(0,0)]\)

```
val a1 = ____________________  
val b1 = ____________________  
val c1 = f1(a1,b1)  
val d1 = f2(a1,b1)  
```

(b) (5 points) Fill in the blanks so that d2 but not c2 is bound to \([(2,2),(1,1),(0,0)]\)

```
val a2 = ____________________  
val b2 = ____________________  
val c2 = f1(a2,b2)  
val d2 = f2(a2,b2)  
```

(c) (5 points) Fill in the blanks so that c3 but not d3 is bound to \([(2,2),(1,1),(0,0)]\)

```
val a3 = ____________________  
val b3 = ____________________  
val c3 = f1(a3,b3)  
val d3 = f2(a3,b3)  
```

**Solution:**

(a) a1 and b1 must both be \([2,1,0]\).
(b) One of a2 and b2 must be \([2,1,0]\) and the other must be \([2,1]\).
(c) One of a3 and b3 must be \([2,1,0]\) and the other must have at least 4 elements and start with \([2,1,0]\).
3. For each of the following programs, give the value that ans is bound to after evaluation:

(a) (4 points)

```ml
val x = 1
fun f y = 
    let
        val x = y + 1
        val y = x + 1
    in
        y + 1
    end
val z = f 4
fun f x = x
val ans = z
```

(b) (4 points)

```ml
val x = 1
val y = 2
fun f (g,h) = g x + h y
val x = 3
val y = 4
val ans = f ((fn z => x), (fn z => z))
```

(c) (4 points)

```ml
exception E
val x = 1
fun f x = if x=2 then raise E else 14
val x = 2
val ans = ((f x) + 4) handle E => 9
```

(d) (4 points)

```ml
val z = 2
val f = (fn x => x + 1) o (fn y => if y=z then 4 else y)
val z = 3
val ans = List.map f [1,2,3,4,5]
```

Solution:

(a) 7
(b) 5
(c) 9
(d) [2,5,4,5,6]
4. (a) (10 points) Without using any helper functions (such as \texttt{fold1}), write an ML function \texttt{in\_order} that behaves as follows:

- It takes two arguments \textit{in curried form}: (1) a function \( f \) that given a list element produces an integer and (2) a list \( \textit{xs} \).
- It returns true if and only if for all elements of \( \textit{xs} \), \( f \) applied to the element returns a number less than or equal to \( f \) applied to any later elements of the list. (This means the result is true for any list with fewer than two elements.)

(b) (6 points) Using \texttt{in\_order}, write a function \texttt{shorter\_strings} that takes a list of strings and returns true if and only if each string in the list is longer than the strings that come later in the list. Hint: You can use ML’s \( \sim \) operator for negation.

c) (4 points) What is the type of \texttt{in\_order}?

d) (2 points) What is the type of \texttt{shorter\_strings}?

e) (4 points) When your solution to part (a) is given a list \( \textit{xs} \) of length \( n \), how many times is the function passed for \( f \) called before \texttt{in\_order} returns?

f) (3 points) Suppose another student has a different answer to part (e) and you are both correct because you have different correct answers to part (a). Are your solutions to part (a) equivalent? Explain briefly.

\textbf{Solution:}

(a) This solution is probably the easiest, but arguably not as good as one that calls \( f \) once for each list element.

\begin{verbatim}
fun in_order f xs =
  case xs of
    [] => true
  | [ ] => true
  | head::neck::tail => f head <= f neck andalso in_order f (neck::tail)
\end{verbatim}

(b) \textit{This question was badly worded: It should have said longer or the same length as, but almost everyone attempted it as intended.}

\begin{verbatim}
val shorter_strings = in_order (fn s => ~ (String.size s))
\end{verbatim}

c) (\texttt{\texttt{a -> int}}) \rightarrow (\texttt{\texttt{a list}}) \rightarrow \texttt{bool}

d) \texttt{string list \rightarrow bool}

e) \textit{This question was not worded well. We meant to ask the number of times called when in\_order returns true. Most people answered it that way. For the intended question and the answer to part (a) above, \( 2n - 2 \), but it depends on how part (a) is written.}

f) No, because if \( f \) has any side-effects (e.g., printing or assigning to mutable data), then the two functions could behave differently. But if \( f \) is a “pure function” then the answer is yes.
5. In this problem, suppose we have an ML structure $M$ and signature $S$ in this standard usage:

```ml
signature S =
  sig
    ...
  end
structure M :> S =
  struct
    ...
  end
```

Assume everything type-checks initially, meaning $M$ matches $S$. For each of the following statements, answer “always,” “sometimes,” or “never.”

(16 points) (2 points each)

(a) If $S$ originally contains `val f : int -> int` and we comment out this line, then $M$ will still match $S$.

(b) If $S$ originally contains `val f : int -> int` and we comment out this line, then a client of $M$ will still type-check.

(c) If $S$ originally does not contain `val g : string -> string` and we add it to $S$, then $M$ will still match $S$.

(d) If $S$ originally does not contain `val g : string -> string` and we add it to $S$, then a client of $M$ will still type-check.

(e) If $S$ originally contains an abstract type `type t` and we replace this line with `datatype t = Foo of int | Bar of bool`, then $M$ will still match $S$.

(f) If $S$ originally contains an abstract type `type t` and we replace this line with `datatype t = Foo of int | Bar of bool`, then a client of $M$ will still type-check.

(g) If $S$ originally contains the line `datatype t = Foo of int | Bar of bool`, and we replace this line with `type t`, then $M$ will still match $S$.

(h) If $S$ originally contains the line `datatype t = Foo of int | Bar of bool`, and we replace this line with `type t`, then a client of $M$ will still type-check.

**Solution:**
Explanations were not required, but are included here.

(a) Always: If $M$ matches everything in $S$, it will still match with one less variable binding.

(b) Sometimes: A client will type-check if and only if it was not using $M.f$.

(c) Sometimes: It will match if and only if defines a function $g$ with a type equal or more general than `string->string`.

(d) Always: Providing another function outside the module cannot cause code not to type-check – it just was not using this feature before. (Will also accept answer Sometimes if justified in terms of the open construct and shadowing.)

(e) Sometimes: It will match if and only if its internal definition of type $t$ is this datatype binding.

(f) Always: The client type-checked without knowing the representation of $M.t$, so it will still type-check without using this extra knowledge.

(g) Always: We can take any type we were exposing concretely and hide it via a signature.

(h) Sometimes: A client will type-check if and only if it was not using any of $t$’s constructors – either as functions or as patterns.
Name:__________________________________________

More room in case you need it.