Name __________________________________  Section ________________

Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-notes
- Please stop promptly at 10:20
- There are a total of 52 points, distributed unevenly among the questions
- Please try to write neatly – style matters, but we’ll take into account the fact that this is a short exam and it’s not always possible to have the time to revise and clean up everything.

Advice:

- Read questions carefully and understand what’s asked before you start writing.
- Leave evidence of thoughts and intermediate steps so you can get partial credit.
- Skip around – if you get hung up on a question, try the next one and come back.
- If you have questions, ask – raise your hand and someone will try to help you out.
- Relax. You are here to learn.
Question 1. (8 points) What are the types of the following function definitions?

(a) fun clone x = (x, x);

(b) fun fst(x, y) = x;

(c) fun ffst z = fst (fst z);

(d) fun g (x, y, z) = x (y z);

Question 2. (8 points) Write a tail-recursive function \texttt{len lst} that calculates the length of the list \texttt{lst}. For example, \texttt{len[]} should evaluate to 0, \texttt{len[1,2,3,4]} should evaluate to 4, \texttt{len[[1,2,3], 4]} should evaluate to 2. For full credit your solution must use pattern matching, not the \texttt{hd} and \texttt{tl} functions or if-statements. Also, if your solution involves an auxiliary, or helper function, that function should be defined locally in \texttt{len} and not defined externally as a top-level function.
Question 3. (3 points) SML provides a lot of “syntactic sugar” to make it possible to use convenient notation for more basic underlying constructs. For instance, we can define a tuple e

\[
\text{val } e = (123, 456, 789);
\]

and reference its fields as \#1 e, \#2 e, \#3 e. But this is syntactic sugar for a record datatype. How could you define e if the tuple syntactic sugar were not available?

Question 4. (8 points) Arithmetic expressions involving integers, addition, and multiplication, can be represented as a data structure in an ML program with the following data type.

\[
\text{datatype } \text{expr} = \text{Int of int} \\
| \text{Prod of expr * expr} \\
| \text{Sum of expr * expr}
\]

Write a recursive function eval e:expr that, given an expression e, evaluates the expression and returns its value.
Question 5. (6 points) For each of the following sets of expressions and definitions, write the value of the final expression.

(a) val k = 17;
    fun f k = k+1;
    fun g n = f k;
    val k = 42;
    g(k+1);

(b) val n = 2;
    fun f x = let val y = x+1 in fn g => n+y end;
    fun g x = f 4;
    g 1 2;

Question 6. (8 points) Write a curried function head that has two parameters, an integer \( k \) and a list \( lst \). The result of executing head \( k \) \( lst \) should be a list consisting of the first \( k \) items in \( lst \). For example, head 3 [1,2,3,4,5] should evaluate to [1,2,3]. The result of evaluating head \( k \) should be a function that, when applied to a list, yields the first \( k \) items in the list. So, for example, if the result of head 3 is applied to the list [1,2,3,4,5], it should evaluate to [1,2,3]. If the list has fewer than \( k \) elements, the function head \( k \) (or head \( k \) \( lst \)) should generate a TooFewElements exception.

    exception TooFewElements;

    fun head k lst =
Question 7. (3 points) Both of the following signatures define the interface to a complex number structure. What’s the significant difference between them from the perspective of a programmer using these signatures?

signature COMPLEX_A =
  sig
    datatype complex = Pair of real * real | Real of real
    val make_complex : real * real -> complex
    val add : complex * complex -> complex
    val print_complex : complex -> unit
  end

signature COMPLEX_B =
  sig
    datatype complex
    val make_complex : real * real -> complex
    val add : complex * complex -> complex
    val print_complex : complex -> unit
  end
Question 8. (8 points) The ML standard library provides several higher-order functions for manipulating lists, in particular `map`, `filter`, `foldl` (fold left), and `foldr` (fold right). These are defined as follows:

\[
\text{map } f \ [x_1, \ldots, x_n] = [f \ x_1, \ldots, f \ x_n]
\]

\[
\text{filter } f \ [x_1, \ldots, x_n] = \text{a list containing all elements } x_i \text{ in the original list where } f \ x_i \text{ evaluates to true}
\]

\[
\text{foldl } f \ e \ [x_1, \ldots, x_n] = f(x_n, \ldots, f(x_1, e)\ldots)
\]

\[
\text{foldr } f \ e \ [x_1, \ldots, x_n] = f(x_1, \ldots, f(x_n, e)\ldots)
\]

The fold functions apply the function \( f \) to the list elements from left to right (\( \text{foldl} \)) or right to left (\( \text{foldr} \)) to produce a single result.

(a) What are the types of these functions?

\[
\text{map}
\]

\[
\text{foldl}
\]

(b) Use some combination of these functions and any anonymous functions you need to define a function `sumpos` that returns the sum of all the positive numbers in a list of integers, for example, `sumpos \ [3, -4, 12, 0, 5]` would evaluate to 20. You can assume that the list has type `int list` (i.e., it only contains integers). You should not use any loops or recursion in your solution – just use some combination of the higher-order functions to calculate the result – and you should not define (bind) any other top-level functions other than `sumpos`. 