Please do not turn the page until the bell rings.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- **Please stop promptly at 3:20.**
- 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, which describes “expression trees” where leaves are constants or variables and internal nodes are additions or multiplications.

    datatype exp = Constant of int
               | Variable of string
               | Add of exp * exp
               | Multiply of exp * exp

(a) (10 points) Write an ML function has_variable of type exp * string -> bool that returns true if and only if the string appears somewhere in the expression. You can use = to compare strings.

(b) (10 points) Write an ML function const_not_under_add of type exp -> bool that returns true if and only if there exists at least one constant that is not “underneath” at least one addition. For example, Multiply(Add(Constant 3, Constant 4), Multiply(Variable "x", Constant 6)) would produce true because the 6 is not under an addition, but Add(Multiply(Constant 1, Constant 2), Multiply(Constant 3, Constant 4)) would produce false because all the constants are under an addition even though they are not “directly” under it.
2. This problem considers this ML code:

```ml
exception BadArgs

fun f (g, xs, ys) =
  case (xs, ys) of
    ([], []) => true
  | (x::xs, y::ys) => f(g, xs, ys) andalso g(x, y)
  | _ => raise BadArgs
```

(a) (12 points) For each of the following expressions that use `f`, fill in the blank with an argument that causes the overall expression to evaluate to `true`.

i. `f((fn (x, y) => x > y), [3, 4, 5], ____________)`
ii. `not (f((fn (x, y) => x > y), [3, 4, 5], ____________))`
iii. `((f ((fn (x, y) => x > y), [3, 4, 5], ____________)) ; false) handle BadArgs => true`

(b) (5 points) In English, briefly describe what `f` computes (not how it computes it).

(c) (4 points) Is `f` tail-recursive? Explain briefly.
3. For each of the following programs, give the value that \texttt{ans} is bound to after evaluation.

(a) (5 points)

\begin{verbatim}
val x = 2
val y = 3
fun f z = 
  let
    val y = x
    val x = y
  in
    x + y + z
  end
val z = 4
val ans = f x
\end{verbatim}

(b) (6 points)

\begin{verbatim}
val x = 1
fun f y = 
  if y > 2
    then fn z => x + z
    else fn z => x - z
val x = 3
val g = f 4
val x = 5
val ans = g 6
\end{verbatim}

(c) (5 points)

\begin{verbatim}
fun f x = 
  case x of
    [] => 0
  | (a,b)::[] => a + b
  | (a,b)::(c,d)::_ => a + d
val ans = f (List.map (fn x => (1,x)) [2,4,8,16,32])
\end{verbatim}
4. (a) (10 points) Without using any helper functions, write an ML function `filter_map`, which combines aspects of `List.filter` and `List.map`, as follows:

- It takes two arguments in curried form: (1) a function `f` that takes list elements and produces options and (2) a list `xs`.
- It returns a list.
- If `v1` is a value in the input list and `f v1` returns `SOME v2`, then `v2` is in the output list.
- Like `List.map`, it preserves the order of results: if `v1` precedes `v2` in the input, `f v1` is `SOME v3`, and `f v2` is `SOME v4`, then `v3` precedes `v4` in the output.
- Like `List.filter`, the result list may be shorter than the input list: if `f` returns `NONE` for `n` elements, then the result will have `n` fewer elements.

(b) (4 points) What is the type of `filter_map`?

(c) (6 points) Use a `val` binding and `filter_map` to define `positive_lengths`, which should take a list of strings and return the lengths of all non-empty strings. For example,

```ml
positive_lengths ["", "hi", "currying", ",", ",", "341"] evaluates to [2,8,3].
```

Use `String.size` as part of your solution.

(d) (2 points) What is the type of `positive_lengths`?

(e) (2 points) Here is an alternate implementation of `filter_map` if you fill in the blanks with the right ML library functions. Do so.

```ml
fun filter_map f = (___________ valOf) o (___________ isSome) o (___________ f)
```
5. *This problem continues onto the next page and has a part (b).*

Consider this structure definition:

```ml
structure NonEmptyStringList :> NESTRINGLIST =
  struct
  type t = string list
  fun newList s = [s]
  fun cons (s,ss) = s::ss
  fun longest ss =
    case ss of
      [] => raise List.Empty
    | [s] => s
    | s::ss => if String.size s >= String.size(longest ss) then s else longest ss
  end
```

(a) *(16 points)* For each of the five following definitions of `NESTRINGLIST`, decide which of the following is true for client code (code outside the module) and *briefly justify your choice*:

- A: It can cause an exception by calling `NonEmptyStringList.longest` with an empty list.
- B: It cannot call `NonEmptyStringList.longest` at all.
- C: It can call `NonEmptyStringList.longest`, but not in a way that can cause an exception.

```ml
signature NESTRINGLIST =
  sig
  type t = string list
  val newList : string -> t
  val cons : string * t -> t
  val longest : t -> string
  end
```

```ml
signature NESTRINGLIST =
  sig
  type t = string list
  val cons : string * t -> t
  val longest : t -> string
  end
```

```ml
signature NESTRINGLIST =
  sig
  type t = string list
  val newList : string -> t
  val cons : string * t -> t
  val longest : t -> string
  end
```
signature NESTRINGLIST =
  sig
  type t
  val newList : string -> t
  val cons : string * t -> t
  val longest : t -> string
  end

signature NESTRINGLIST =
  sig
  type t
  val newList : string -> string list
  val cons : string * t -> t
  val longest : t -> string
  end

(b) (3 points) Even for the signature(s) where the answer is (C), why is longest a very poorly written function? Describe an argument that a client could create for which longest would perform very badly.
Name:__________________________________

This page intentionally blank.