• Do not start the exam until you are told to.

• This is a open-book, open-notes exam, but with no computational devices allowed (such as calculators/cellphones/laptops).

• Do not look at anyone else’s exam. Do not talk to anyone but an exam proctor during the exam.

• Write your answers in the space provided.

• Wherever it gives a line limit for your answer, write no more than the specified number of lines. The rest will be ignored.

• Work out your solution in blank space or scratch paper, and only put your answer in the answer blank given.

• In all exercises, you are allowed to use the “@” operator.

• Good luck!
1. [20 points] Consider the following data type for representing arithmetic expressions with constants, variables and binary operators:

```ocaml
type expr =  
  | Const of int  
  | Var of string  
  | Op of string * expr * expr;;
```

For example:

- `Op ("+", Var "a", Const 4)` represents `a+4`
- `Op ("+", Var "a", Op ("-", Var "b", Const 4))` represents `a+(b-4)`

a. [10 points] You will write a function `rename_var : expr -> string -> string -> expr` which renames variables. In particular, given an expression `e`, and two variable names `n1` and `n2`, `(rename_var e n1 n2)` returns a new expression in which all occurrences of variable `n1` have been replaced with variable `n2`. For example:

```ocaml
# rename_var (Op ("+", Var "a", Const 4)) "a" "b";;
- : expr = Op ("+", Var "b", Const 4)

# rename_var (Op ("+", Var "a", Const 4)) "b" "c";;
- : expr = Op ("+", Var "a", Const 4)

# rename_var (Op ("+", Op ("*", Var "x", Var "y"), Op ("-", Var "x", Var "z"))) "x" "y";;
```

Fill in the code below for `rename_var`:

```ocaml
let rec rename_var e n1 n2 =

```

```ocaml```

---

2.
b. [ 10 points ] You will write a function \texttt{to\_str : expr -> string} which takes an expression and returns a string representation of that expression. For example:

\begin{verbatim}
# to_str (Op (+, Var "a", Const 4));;
- : string = "a+4"

# to_str (Op (+, Const 10, Op (+, Const 10, Var "b")));;
- : string = "10+(10+b)"

# to_str (Op (+, Op (*, Var "x", Var "y"), Op (-, Var "x", Var "z")));;
- : string = "(x*y)+(x-z)"
\end{verbatim}

Carefully note the behavior of parentheses: parentheses are added around binary expressions, except if the expression is at the top-level, in which case no parentheses are added. In particular, the nested expressions above, like “10+b” and “x*y”, have parentheses around them, but “a+4” does not have parentheses, because the expression is at the top-level.

Fill in the code for \texttt{to\_str} below. You will want to make use of the built-in OCaml function \texttt{string\_of\_int : int -> string} which converts an integer to its string representation, and the \texttt{^} operator which concatenates two strings.

\begin{verbatim}
let to_str e =

  let rec str_helper e top_level =

    ...

    ...

    ...

    ...

    ...

    ...

  ...

  in

  str_helper e true;;
\end{verbatim}
Scratch space to work out your answers:
2. [20 points] You will use `fold_left` to write a function `average_if : (int -> bool) -> int list -> int`. Given a “tester” function \( f \) and a list \( l \) of integers, `average_if \( f \) \( l \)` returns the average of all integers in \( l \) for which \( f \) returns true, or 0 if \( f \) returns false for all integers in \( l \). For example:

```ocaml
# let even x = x mod 2 = 0;;
val even : int -> bool = <fun>

# average_if even [1;2;3;4;5];; (* returns average of 2,4 to 3*)
- : int = 3

# average_if even [1;2;3;4;5;6;7;8];; (* returns average of 2,4,6,8 to 5 *)
- : int = 5

# average_if even [1;3;5;7];; (* no even numbers to 0 *)
- : int = 0
```

Fill in the implementation of `average_if` below. Recall that the type of `fold_left` is:

`fold_left: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`

**Hint:** The accumulator is a pair.

```ocaml
let average_if f l =
  let folding_fn _____________________ _____________________ =
  ___________________________________
  ___________________________________
  ___________________________________
  ___________________________________
  ___________________________________

  let base = _____________________ in

  let _____________________ = List.fold_left folding_fn base l in
  ___________________________________
  ___________________________________
```
Scratch space to work out your answers:
3. [10 points]

a. [5 points] You will use map and fold_left to write a function length_2 : int list list -> int, which takes a list of lists of integers, and returns the total number of integers in all the lists. For example:

```ml
# length_2 [[1;2;3];[4;6]];;
- : int = 5
```
```ml
# length_2 [[1;2;3];[4;6];[9;10]];;
- : int = 7
```
```ml
# length_2 [[];[];[]];;
- : int = 0
```

Recall that the type of fold and map are:

- fold_left: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
- map: ('a -> 'b) -> 'a list -> 'b list

You can also use the length function, which has type 'a list -> int.

Fill in the implementation of length_2 below.

```ml
let length_2 l =

List.fold_left (+) ________________ (List.map ________________ ________________ )
```

b. [5 points] You will now use map, fold_left and length_2 to write a function length_3 : int list list list -> int, which takes a list of lists of lists of integers (wow!), and returns the total number of integers in all the lists. For example:

```ml
# length_3 [[[1;2;3]];[[4;6];[7;8]]];;
- : int = 7
```
```ml
# length_3 [[[1;2;3]];[[4;6];[7;8];[10;11]]];;
- : int = 9
```

Fill in the implementation of length_3 below.

```ml
let length_3 l =

List.fold_left (+) ________________ (List.map ________________ ________________ )
```
4. [ 10 points ] For each expression below, write down the returned value (not the type).

let f1 = List.map (fun x->2*x);;

f1 [1;2;3;4];; ----------------------------------------

let f2 = List.fold_left (fun x y -> (y+2)::x) [];;

f2 [3;5;7;9];; ----------------------------------------

let f3 = List.fold_left (fun x y -> x@[3*y]) [];;

f3 [1;3;6];; ----------------------------------------

(* This is going to get harder now... *)

let f = List.fold_left (fun x y -> y x);;

f 1 [(+) 1; (-) 2];; ----------------------------------------

f "abc" [(^) "zzz"; (^) "yyy"];; ----------------------------------------

(* Ok, this one is insanely hard!!! *)

f [1;2;3] [f1;f2;f3];; ------------------------------------------