Before reading further, please arrange to have an empty seat on either side of you. Now that you are seated, please write your name on the back of this exam.

This is a closed-notes and closed-book exam. Computers, calculators, and books are prohibited.

This is a 12 point exam. Answer all questions in Section 1 and question 2.1. Answer either 2.2 or 2.3 but not both. Circle the number of the problem that you want graded.

• Partial credit will be given so be sure to show your work.

• Feel free to write helper functions if you need them.

• Please write neatly.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Out Of</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>Required</td>
</tr>
<tr>
<td>2.1</td>
<td>3</td>
<td>Required</td>
</tr>
<tr>
<td>2.2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>
Section 1 (5 Points Total)

1. (1 Point) In a sentence, what is the significance of the number 256 in computing?

   Answer: $2^8 = 256$, there are 256 8-bit patterns.

2. (1 Point) Is the following well-defined? If so, what is its value?

   let a = let a = 2 in a * a in a * a

   Answer: Yes.

   let a = let a = 2 in a * a in a * a
   \(\rightarrow\) let a = 2 * 2 in a * a
   \(\rightarrow\) let a = 4 in a * a
   \(\rightarrow\) 4 * 4
   \(\rightarrow\) 16

3. (1 Point) Solve for \(X\).

   (a) \(66_{10} = X_{16}\).

      Answer: \(X = 42\).

   (b) \(11011_2 = X_4\).

      Answer: \(X = 123\).
4. (2 Points) The `List.tl : 'a list -> 'a list` function returns the tail of a list. For example, the call `(List.tl [1; 2; 3])` evaluates to the list `[2; 3]`. Given the call, `(f [1; 2; 3])`, show the state of the stack and heap after (1) has executed but before (2) has executed.

```ocaml
let f xs =  
  let ys = List.tl xs in 
  let zs = (xs, ys) in 
  zs

let f [1; 2; 3]
```

**Answer:**

```
+----+ | f  | +----+
|     +-----------------------------+  
|     | +---------+ +---------+ +---------+  
|     | o---------| 1  | o---------| 2  | o---------| 3  | o---------|=  
|     +---------+ +---------+ +---------+ +---------+ =  
|     | +---------| +----|----| +----|----| +----|----|  
|     | o---------| |     | |     | |     | |     | |     |  
|     +---------+ +----|----+ +----|----+ +----|----+ +----|----+  
| +------|  
| o-+----+  
+-------+  
```

Section 2 (7 Points Total)

Do problem 1 and either 2 or 3 but not both. Circle the number of the problem that you want graded.

1. (3 Points) In the run length encoding system, sequences of repeated values \( v \ v \ldots \ v \) can be represented efficiently by pairs \((v, n)\) where \( n \) is the length of the sequence. For example, a run length encoding would represent the sequence of bits \([0; 0; 0; 1; 0; 0; 0; 0]\) as the list of pairs \([(0, 3); (1, 2); (0, 4)]\). Write the expand function \( \text{expandRLE} : ('a \times \text{int}) \text{list} \to 'a \text{ list} \) such that a call \( \text{expandRLE} \ rle \) evaluates to the expanded list.

Answer:

\[
\text{let rec copy item n = }
\text{ \quad match n = 0 with}
\text{ \quad \quad | true \to []}
\text{ \quad \quad | false \to item::(copy item (n - 1))}
\]

\[
(* \text{expandRLE : ('a \times \text{int}) list \to 'a list} *)
\]

\[
\text{let rec expandRLE rle =}
\text{ \quad match rle with}
\text{ \quad \quad | [] \to []}
\text{ \quad \quad | (item, n)::rest \to (copy item n) \text{@} (expandRLE rest)}
\]

2. (4 Points) Write a function \( \text{product} : 'a \text{ list} \to 'b \text{ list} \to ('a \times 'b) \text{ list} \) such that a given call \( \text{product} \ xs \ ys \) returns a list of all pairs formed from elements of \( xs \) and \( ys \). For example, the call \( \text{product} \ [1; 2] ['A'; 'B'] \) should evaluate to \([(1, 'A'); (1, 'B'); (2, 'A'); (2, 'B')]\).

Answer:

\[
\text{let rec product xs ys =}
\text{ \quad let nested = List.map (fun x \to List.map (fun y \to (x, y)) ys) xs}
\text{ \quad in}
\text{ \quad List.fold_left (@) [] nested}
\]
3. (4 Points) Write a function `merge : int array -> int array -> int array` such that a given call `(merge a b)`, where `a` and `b` are arrays of integers in ascending order, returns an integer array containing the values in `a` and `b` in ascending order. For example, the call `(merge [|1; 3; 5|] [|2; 4|])` should return the 5-element array [|1; 2; 3; 4; 5|].

Answer:

```ocaml
let merge a b =
  let na = Array.length a in
  let nb = Array.length b in
  let c = Array.make (na + nb) 0 in
  let rec repeat ai bi =
    match (ai < na, bi < nb) with
    | (true, true) -> if a.(ai) < b.(bi) then
      begin
        c.(ai+bi) <- a.(ai);
        repeat (ai+1) bi
      end
    else
      begin
        c.(ai+bi) <- b.(bi);
        repeat ai (bi+1)
      end
    | (true, false) -> c.(ai+bi) <- a.(ai);
    | (false, true) -> c.(ai+bi) <- b.(bi);
    | (false, false) -> ()
  in
  repeat 0 0;
  c

Or, cutting corners:

let merge a b =
  Array.of_list (List.merge compare (Array.to_list a) (Array.to_list b))
```