1. (6 pts) OCaml Types and Type Inference
   a. (2 pts) Give the type of the following OCaml expression
      \[ \text{fun} \ x \rightarrow (x, 2) \]
      \[ \text{Type} = ('a -> ('a * int)) \]
   b. (2 pts) Write an OCaml expression with the following type
      \[ \text{int} 
      \rightarrow \text{(float * int list)} \]
      \[ \text{Code} = \]
      \[ \text{Example solutions:} \]
      \[ \text{let f x = (2.0, [x+3]) etc...} \]
   c. (2 pts) Give the value of the following OCaml expression. If an error exists, describe the error.
      \[ \text{(fun z -> fun y -> z - y)} 5 \ 3 \]
      \[ \text{Value} = 2 \]

2. (8 pts) OCaml Programming
   Using the following code for either map/fold and an anonymous function, write a function getFirsts which given a list of pairs, returns a list of the 1\textsuperscript{st} members of each pair as a list (in original or reverse order). Partial credit given for solutions which do not use map/fold.
   
   Example: getFirsts [(1,2);(3,4);(3,5)] = [1;3;3] OR [3;3;1]
   getFirsts [("a","x");("b","y");("c","z")] = ["a";"b";"c"] OR ["c";"b";"a"]
   
   \[
   \begin{array}{|c|c|}
   \hline
   \text{let rec map f l = match l with} & \text{let rec fold f a l = match l with} \\
   [] -> [] & [] -> a \\
   | (h::t) -> (f h)::(map f t) & | (h::t) -> fold f (f a h) t \\
   \hline
   \end{array}
   \]
   
   \[ \text{let getFirsts lst = map (fun a -> match a with (h,t) -> h) lst} \]
   \[ \text{OR} \]
   \[ \text{let getFirsts lst = fold (fun a y -> match y with (h,t) -> h::a) [] lst} \]
   \[ \text{OR} \]
   \[ \text{let getFirsts lst = map (fun (h,t) -> h) lst} \]
   \[ \text{OR} \]
   \[ \text{let getFirsts lst = fold (fun a (h,t) -> h::a) [] lst} \]
   \[ \text{etc...} \]

3. (6 pts) Context free grammars
   Consider the following grammar: \[ S \rightarrow aSaS \mid e \text{ (* epsilon *)} \]
   a. (2 pts) Describe the set of strings generated by the grammar.
      Strings of even numbers of a’s. I.e., (aa)*
   b. (4 pts) Is the grammar ambiguous? Show proof if possible.
      Grammar is ambiguous since there are 2 leftmost derivations for “aaaa”.
      \[ S \Rightarrow aSaS \Rightarrow aaSaS \Rightarrow aaaaS \Rightarrow aaaa \]
      \[ S \Rightarrow aSaS \Rightarrow aaSaSaS \Rightarrow aaaaS \Rightarrow aaaa \]