1. (8 pts) OCaml Types and Type Inference
   
   a. (2 pts) Give the type of the following OCaml expression

   \[
   \text{fun } x \ y \rightarrow (y + 2, x)
   \]

   \textbf{Type =}

   \[
   \text{fun } x \ y \rightarrow (y + 2, x)
   \]

   b. (3 pts) Write an OCaml expression with the following type

   \[
   (\text{float list } \rightarrow \text{float } \rightarrow 'a) \rightarrow 'a
   \]

   \textbf{Code =}

   \[
   \text{fun } x \ y \rightarrow (y + 2, x)
   \]

   c. (3 pts) Give the value of the following OCaml expression. If an error exists, describe the error. The function fold is given for problem 2.

   \[
   \text{fold } ((\text{fun } x \ y \ z \rightarrow x + (y * z)) \ 2) \ 1 \ [1; \ 2; \ 3];
   \]

   \textbf{Value =}
2. (8 pts) OCaml Programming

Using either map or fold and an anonymous function, write a curried function called *divisible* which when given a number *n* and a list of ints *lst*, returns a list of all elements of *lst* that are divisible by *n* (maintaining their relative ordering). You are allowed to use List.rev (reverses a list) and the (curried) map and fold functions provided, but no other OCaml library functions. **Hint:** *x* is divisible by *y* iff \((x \mod y = 0)\) is true.

```ocaml
let rec map f l = match l with
  | [] -> []
  | (h::t) -> (f h)::(map f t)
let rec fold f a l = match l with
  | [] -> a
  | (h::t) -> fold f (f a h) t
```

Example:

```ocaml
divisible 4 [3;16;24] // returns [16; 24]
divisible 3 [4;1;11]  // returns []
divisible 3 []        // returns []
```

3. (4 pts) Context Free Grammars

Consider the following grammar:

\[
S \rightarrow aSc \mid b \mid \epsilon
\]

a. (2 pts) Describe the set of strings accepted by this grammar.

b. (2 pts) Draw a parse tree for the string aabcc.