1. (16 pts) OCaml Types and Type Inference
   a. (2 pts each) Give the type of the following OCaml expressions
      i. [ (“1”, 2) ; (“3”, 4) ]  \( \text{Type} = (\text{string} \times \text{int}) \text{ list} \)
      ii. \( \text{fun f a -> [a ; a+1]} \)  \( \text{Type} = \text{int} \rightarrow \text{int list} \)
   b. (3 pts each) Write an OCaml expression with the following type
      i. \( \text{int} \times \text{int list} \)  \( \text{Code} = (2, [2]) \)
      ii. \( \text{int list} \rightarrow (\text{int} \rightarrow \text{int}) \)  \( \text{Code} = \text{let f a b = match a with (h::t) -> h+b} \)
   c. (3 pts each) Give the value of the following OCaml expressions. If an error exists, describe the error.
      i. \([1;2];[3]\)
         \( \text{Error} = 3 \text{ has type } \text{int} \text{ but is used with type } \text{int list} \)
         OR \( [3] \text{ has type } \text{int list} \text{ but is used with type } \text{int list list} \)
         OR can only add \( \text{int list} [1;2] \) to \( \text{int list list} \)
         OR trying to add \( \text{int list} [1;2] \) to \( \text{int list} [3] \)
      ii. \( \text{let x y = y 3 in x (fun z -> z – 1)} \)
         \( \text{Value} = 2 \)

2. (18 pts) OCaml Programming
   Solve the following OCaml programming problems. You are allowed to use \text{List.rev} (reverses a list) and the following (curried) map and fold functions, but no other OCaml library functions. Your solution must run in \( O(n) \) time for input lists of length \( n \).
   a. (9 pts) Write a function \text{makeLists} which when applied to a list \( \text{lst} \), creates a new list for every element of \( \text{lst} \), returning the results in a single list. You may use map or fold if you wish, but it is not required.
      Example: \( \text{makeLists} [1;2;4] = [[1];[2];[4]] \)
      Some possible answers:
      \( \text{let rec makeList x = match x with} \)
      \( \text{[] -> []} \)
      \( ! (h::t) -> [h]::(\text{makeList t}) ;; \)
      \( \text{let makeList x = map (fun y -> [y]) x ;;} \)
      \( \text{let makeList x = map (fun y -> y::[ ])} x ;; \)
      \( \text{let makeList x = List.rev (fold (fun a y -> [y]::a) [ ] x) ;;} \)
b. (9 pts) Using either map or fold and an anonymous function, write a function `over20` which when applied to a list of ints `lst`, returns a list of all elements of `lst` that are 21 or over (preserving their relative order in `lst`).

Example: over20 [33;18;21;19] = [33;21]

```ml
let over20 x = List.rev (fold (fun a y -> if (y > 20) then y::a else a) [] x) ;;
```

Partial credit:

```ml
let rec over20 x = match x with [] -> [] | (h::t) -> if (h > 20) then (h::(over20 t)) else (over20 t) ;;
```

3. (18 pts) Context Free Grammars

Consider the following grammar: $S \rightarrow aA \mid A \ A \rightarrow bS \mid ca$

a. (8 pts) What is the set of strings generated by this grammar?

$\mathcal{L}(a \mid \epsilon)(b\mid ba)^*ca$

b. (10 pts) Write a derivation of the string “ababca”

$S \rightarrow aA \rightarrow abS \rightarrow abaA \rightarrow ababS \rightarrow ababA \rightarrow ababca$