Name ________________________________

<table>
<thead>
<tr>
<th>Discussion Time</th>
<th>10am</th>
<th>11am</th>
<th>noon</th>
<th>1pm</th>
<th>2pm</th>
<th>3pm</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA Name (circle):</td>
<td>Casey</td>
<td>Casey</td>
<td>Xuefang</td>
<td>Xuefang</td>
<td>Ian</td>
<td>Ian</td>
</tr>
<tr>
<td>(for picking up graded quiz)</td>
<td>Ilse</td>
<td>Daniel</td>
<td>Yogarshi</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Instructions**
- Do not start this test until you are told to do so! You have 20 minutes for this quiz.
- This is a closed book exam. No notes or other aids are allowed.
- For partial credit, write neatly, show all of your work, and clearly indicate your answers.

1. **(12 pts) OCaml programming**

   Solve the following OCaml programming problems. You may use List.rev (reverses a list), the (curried) map and fold functions, and functions from Pervasives (except fst & snd), but no other OCaml library functions. You are not allowed to use imperative OCaml features such as int ref. Your solution must run in O(n) time for input of length n for full credit.

   a. **(6 pts)** Using either map or fold and an anonymous function, write a function `split` that when applied to an associative list `lst` (i.e., list of (key, value) pairs), returns a pair of lists (keys, values), preserving their relative order in `lst`.

   Example: 
   
   ```ocaml
   split [ ] = ( [ ] ; [ ] )
   split [ (1,2) ; (5,6) ; (3,4) ] = ( [1; 5; 3] , [2; 6; 4] )
   split [ (“x”,2.1) ; (“y”,4.3) ] = ( [“x”; “y”] , [2.1; 4.3] )
   ```

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

   ```ocaml
   let rec fold f a l = match l with
   [ ] -> a
   | (h::t) -> fold f (f a h) t
   ```
b. (6 pts) Consider the following OCaml user-variant type `intList` implementing a linked list of ints.

```
type intList =
  | Empty
  | Node of int * intList
```

i. (2 pts) Write an OCaml expression with type `intList` that is equivalent to the list `[1 ; 2]`.

ii. (4 pts) Write a function `freq` of type `(int -> intList -> int)` that takes an integer `n` and an `intList`, and returns the number of times `n` occurs in the `intList`.
2. (8 pts) Context free grammars & parsing
   Consider the following grammar (S = start symbol and terminals = x y z):
   
   S → xA | yz
   A → SA | epsilon

   a. (2 pts) Draw a parse tree for the string “xyz”
S → xA | yz
A → SA | epsilon

b. (3 pts) Calculate FIRST sets for S and A

FIRST(S) = {}  \quad \text{FIRST(A)} = {}  

c. (3 pts) Using pseudocode, write only the parse_A function found in a recursive descent parser for the grammar. You may assume the functions parse_S already exists.

You may use the following utilities:

<table>
<thead>
<tr>
<th>lookahead</th>
<th>Variable holding next terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>match ( x )</td>
<td>Function to match next terminal to x</td>
</tr>
<tr>
<td>error ( )</td>
<td>Reports parse error for input</td>
</tr>
</tbody>
</table>