Sample Questions for Midterm 1 (CS 421 Fall 2011)

On the actual midterm, you will have plenty of space to put your answers. The actual midterm will likely have no more than 9 questions plus one extra credit question. In addition to questions of the kind asked below, you should expect to see questions from your MPs and HW1 on the exams.
Some of these questions may be reused for the exam.

1. Given the following OCAML code:
   ```
   let x = 3;;
   let f y = x + y;;
   let x = 5;;
   let z = f 2;;
   let x = "hi";;
   ```
   What value will **z** have? Will the last declaration (**let x = "hi";**) cause a type error? What is the value of **x** after this code has been executed?
2. What environment is in effect after each declaration in the code in Problem 1?
3. What the effect of each of the following pieces of code?
   a. **(fun x -> (print_string "a"; x + 2)) (print_string "b"; 4);;**
   b. let f = (print_string "a"; fun x -> x + 2) in f (print_string "b"; 4);;
   c. let f = fun g -> (print_string "a"; g 2) in f (fun x -> print_string "b"; 4 + x);;
4. Consider the following two OCaml functions, **loop1** and **loop2**:
   ```
   let rec loop1 () = loop1(); ()
   let rec loop2 () = loop2();;
   val loop1 : unit -> unit = <fun>
   val loop2 : unit -> 'a = <fun>
   ```
   Suppose you were to run **loop1();;** and **loop2();;** in OCaml, (pressing CTRL + C after at least a minute to terminate infinite loops when necessary).
   a. For each program, what behavior would you expect to see?
   b. What is the difference between loop1 and loop2?
   c. For each program state if it is:
      i. recursive,
      ii. forward recursive,
      iii. tail-recursive.
5. Write an OCAML function **pair_up** that takes first a function, then an input list and returns a list of pairs of an element from input list (the second argument), paired with the result of applying the first argument to that element. What is the OCAML type of **pair_up**? What is the result of the following expressions:
   a. **pair_up (fun x -> x + 3) [6;4;1];;**
   b. **pair_up ((fun x -> "Hi, "^x), ["John"; "Mary";"Dana"]);;**
   c. **pair_up (fun x -> x *. 2.0);;**
6. Write an Ocaml function **palindrome :string list -> unit** that first prints the strings in the list from left to right, followed by printing them right to left, recursing over the list only once. (Potential extra credit problem: Do this using each of **List.fold_right** and **List.fold_left** but no explicit use of **let rec**.)

7. Using **fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b**, but without using explicit recursion, write a function **concat** : ‘ a list list -> ‘a list that appends all the lists in the input list of lists, preserving the order of elements. You may use the append function **@.**

8. Write an Ocaml function **list_print : string list -> unit** that prints all the strings in a list from left to right:
    a. using tail recursion, but no higher order functions,
    b. using **fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a** but no explicit recursion.
9. Put the following function in full continuation passing style:
    let rec sum_odd n = if n <= 0 then 0 else ((2 * n) – 1) + sum_odd (n – 1);;
    Primitive operations (+, - , *, <=) do not have to be converted to CPS, but all procedure calls must be.
10. Write the definition of an OCAML variant type **reg_exp** to express abstract syntax trees for regular expressions over a base character set of booleans. Thus, a boolean is a **reg_exp,** epsilon is a **reg_exp**, the concatenation of two **reg_exp**’s is a **reg_exp,** the “choice” of two **reg_exp**’s is a **reg_exp**, and the Kleene star of a **reg_exp** is a **reg_exp**.
11. Given the following OCAML datatype:
    type int_seq = Null | Snoc of (int_seq * int)
    write a tail-recursive function in OCAML **all_pos : int_seq -> bool** that returns **true** if every integer in the input **int_seq** to which **all_pos** is applied is strictly greater than 0 and **false** otherwise. Thus **all_pos (Snoc(Snoc(Snoc(Null,3),5),7))** should returns **true,** but **all_pos (Snoc(Null,~1))** and **all_pos (Snoc(Snoc(Null, 3),0))** should both return **false.**
12. What is **type checking**? What do the terms **static** and **dynamic** mean when referring to type checking? Given an example of a property that can be type checked statically, and an example of a property that can only be checked dynamically.
13. Using the typing rules on the sheet found separately on the exam website, give a full type derivation for the following judgment. You should include labels stating which rules are used where.
    {} |- (let x= true in let f = fun x -> x > 1 in if x && f 3 then 17 else 3 + 5) : int