CS421 Fall 2014 Midterm 1

Name:	
NetID:	

- You have **75 minutes** to complete this exam.
- This is a **closed-book** exam. All materials (e.g., calculators, cell phones and scrap paper), except writing utensils are prohibited.
- Do not share anything with other students. Do not talk to other students. Do not look at another students exam. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.
- If you believe there is an error, or an ambiguous question, you may seek clarification from myself or one of the TAs. You must use a whisper, or write your question out. Speaking out aloud is not allowed.
- Including this cover sheet and rules at the end, there are 16 pages to the exam, including one blank page for workspace. Please verify that you have all 16 pages.
- Please write your name and NetID in the spaces above, and also in the provided space at the top of every sheet.

Question	Points	Bonus Points	Score
1	18	0	
2	10	0	
3	16	0	
4	22	0	
5	14	0	
6	20	0	
7	0	10	
Total:	100	10	

Problem 1. (18 points)

The following are 5 examples of potential sequences of OCaml declarations. Following each is a series of statments. Fill in the circle next to **each true statement**.

(a) (3 points)

```
let x = 2;;
let y = 3;;
let f w z = w + x + z - 5;;
let a = f y;;
let x = 0;;
let b = f 5 3;;
```

◯ This fails to finish compiling with an error at the declaration of `a`

√ **This compiles and assigns `b` a value of 5.**

◯ This compiles and assigns `b` a value of 3.

(b) (6 points)

```
let result =
  let test x = ( print_string "a";
                 fun y -> ( print_string "b"; x ) + ( print_string "c"; y ) )
  in test ( print_string "d"; 4)
          ( test (print_string "e"; 5)  (print_string "f"; 6) );;
```

√ **This assigns `result` a value of 15.**

◯ This prints `abcdef`.

◯ This prints `abcdabcef`.

√ **This prints `feacbdacb`.**

◯ This prints `daeafbcbc`

◯ This prints `fedacbacb`.

(c) (3 points)

```
let a = "hi";;
let f (x, y) = x ^ a ^ y;;
let b = 7;;
let a = b + 2;;
let g x = f "d";;
```

◯ This code will not compile to completion because of a type error in the fourth line.

◯ This codes assigns `g` a function that takes a string and returns the result of prepending `"dhi"` to it.

√ **This will not compile to completion because of a type error in the fifth line.**

(d) (3 points)

```
let x = 3;;
let b = let x = 17 in x + 1;;
let e = x * 2;;
```

results with the bindings:

◯ `b` is bound to 4, `e` is bound to 6.

√ **`b` is bound to 18, `e` is bound to 6.**

◯ `b` is bound to 18, `e` is bound to 34.

(e) (3 points)

```
let f g = (print_string "a"; let x = g() in (print_string "b"; 2+x));;
f (fun () -> print_string "x"; 7);;
```

evaluates to:

√ `axb- : int = 9`

◯ `abx- : int = 9`

◯ `xab- : int = 9`

Problem 2. (10 points)

Consider the following OCaml code. Assume that it is executed in an empty environment. Following the code is a series of statments. Fill in the circle next to each true statement.

```
(* 1 *) let a = 2;;
(* 2 *) let b = 3;;
(* 3 *) let f x y = (a * x) + (b * y);;
(* 4 *) let b = 17;;
(* 5 *) let y = f 1;;
```

√ **The environment after executing the declaration after `(* 1 *)` is**

$$\{\texttt{a} \mapsto 2\}$$

◯ The environment after executing through the declaration after `(* 3 *)` is

$$\{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3;\ \texttt{f} \mapsto \texttt{fun x -> fun y -> (a * x) + (b * y)}\}$$

◯ The environment after executing through the declaration after `(* 3 *)` is

$$\{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3;\ \texttt{f} \mapsto \texttt{fun x -> fun y -> (2 * x) + (3 * y)}\}$$

√ **The environment after executing through the declaration after `(* 3 *)` is**

$$\{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3;\ \texttt{f} \mapsto \langle \texttt{x} \to \texttt{fun y -> (a * x) + (b * y)},\ \{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3\}\rangle\}$$

◯ The environment after executing through the declaration after `(* 3 *)` is

$$\{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3;\ \texttt{f} \mapsto \langle \texttt{x} \to \texttt{y} \to \texttt{(a * x) + (b * y)},\ \{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3\}\rangle\}$$

◯ The environment after executing through the declaration after `(* 4 *)` is

$$\{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3;\ \texttt{f} \mapsto \langle x \to \texttt{fun y -> (a * x) + (b * y)},\ \{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3\}\rangle;\ \texttt{b} \mapsto 17\}$$

◯ The environment after executing through the declaration after `(* 4 *)` is

$$\{\texttt{a} \mapsto 2; \texttt{f} \mapsto \texttt{fun x -> fun y -> (2 * x) + (3 * y)};\ b \mapsto 17\}$$

√ **The environment after executing through the declaration after `(* 4 *)` is**

$$\{\texttt{a} \mapsto 2; \texttt{b} \mapsto 17;\ \texttt{f} \mapsto \langle x \to \texttt{fun y -> (a * x) + (b * y)},\ \{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3\}\rangle\}$$

√ **The value assigned to `y` at the end is**

$$\langle \texttt{y} \to \texttt{(a * x) + (b * y)}, \{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3; \texttt{x} \mapsto 1\}\rangle$$

◯ The value assigned to `y` at the end is

$$\langle \texttt{y} \to \texttt{(2 * x) + (3 * y)}, \{\texttt{a} \mapsto 2; \texttt{b} \mapsto 3; \texttt{x} \mapsto 1\}\rangle$$

Problem 3. (16 points)

(a) (8 points) Write a function `sum_even_squares: int list -> int` that returns the sum of the square of each even number in a list of numbers. If the list contains no even numbers, it should return 0. The only form of recursion you are allowed to use is `tail recursion` and you may not use any library functions (including `@`). You may use `mod` to test if a number is even.

```
# let rec sum_even_squares l = . . . ;;
val sum_even_squares : int list -> int = <fun>
# sum_even_squares [5;2;6;17];;
- : int = 40
```

Solution:

```
let sum_even_squares l =
    let rec sum_even_squares_aux l acc =
        match l with
        | [] -> acc
        | x::xs ->
            if x mod 2 = 0 then
                sum_even_squares_aux xs (x * x + acc)
            else
                sum_even_squares_aux xs acc
    in sum_even_squares_aux l 0
```

(b) (8 points) Write a function `sum_even_squares_op : int -> int -> int` and a value `sum_even_squares_base : int` such that `(List.fold_left sum_even_squares_op sum_even_squares_base) : int list -> int` computes the same function as `sum_even_squares: int list -> int`.

The type of `List.fold_left` is `('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`.

- `let sum_even_squares_op =`

Solution:

```
fun s e -> if e mod 2 = 0 then e * e + s else s
```

- `let sum_even_squares_base =`

Solution:

```
0
```

Problem 4. (22 points)

Below is a list of possible partial or total results for the following evaluation of code in an environment. Please fill in all circles where the entry is equal to the given evalution **via a sequence of rewriting steps**. You are not restricted to equality due to a single step of evalution, but may use as many steps, and in whatever order, you deem appropriate. You may wish to refer to the rewrite rules for evaluaiton on the last page.

Eval(`square_sum x y`, {`x` $\mapsto$ 3; `y` $\mapsto$ 5; `square_sum` $\mapsto$ $\langle$`a` $\to$ `fun b -> a * a + b * b`, { }$\rangle$}) =

For compactness of typesetting, we will define

ρ_0 = {`x` $\mapsto$ 3; `y` $\mapsto$ 5; `square_sum` $\mapsto$ $\langle$`a` $\to$ `fun b -> a * a + b * b`, { }$\rangle$}

◯ Eval(app($\langle$`a` $\to$ `b` $\to$ `a * a + b * b`, { }$\rangle$, 3, 5), ρ_0)

◯ Eval(app($\langle$`a` $\to$ `fun b -> a * a + b * b`, { }$\rangle$, 3, 5), ρ_0)

√ **Eval**(**app**(**Eval**(`square_sum x`, ρ_0), **Eval**(`y`, ρ_0)), ρ_0)

√ **Eval**(**app**(**Eval**(`square_sum x`, ρ_0), 5), ρ_0)

√ **Eval**(**app**(**Eval**(`square_sum x`, ρ_0), $\rho_0(y)$), ρ_0)

√ **Eval**(**app**(**Eval**(**app**(**Eval**(`square_sum`, ρ_0), 3), ρ_0), 5), ρ_0)

√ **Eval**(**app**(**Eval**(**app**($\langle$`a` $\to$ `fun b -> a * a + b * b`, { }$\rangle$, 3), ρ_0), 5), ρ_0)

√ **Eval**(**app**(**Eval**(**app**($\langle$`a` $\to$ `fun b -> a * a + b * b`, { }$\rangle$, 3), ρ_0), 5), ρ_0)

◯ Eval(`a * a + b * b`, $\{a \mapsto 3;\ b \mapsto 5\} + \rho_0$)

√ **Eval**(`a * a + b * b`, $\{a \mapsto 3;\ b \mapsto 5\}$)

√ 34

Problem 5. (14 points)

Consider the following OCaml function:

```
# let rec twist x = if x < 1 then 0 else x - twist (x - 1)
val twist : int -> int = <fun>
```

(a) (4 points) Write the functions `leqk : 'a -> 'a -> ('a -> 'b) -> 'b` (this should have been `leqk : 'a -> 'a -> (bool -> 'b) -> 'b`) and `subk : int -> int -> (int -> 'a) -> 'a` that are the CSP transformation of less than (`<`) and subtraction (`-`).

Solution:

```
let leqk x y k = k (x < y);;
let subk x y k = k (x - y);;
```

(b) (10 points) Write the function `twistk : int -> (int -> 'a) -> 'a` that is the CPS transformation of the above code. Be careful to take note of the type of the function `twistk`, and its arguments. You should use `leqk` and `subk` that you defined above.

Solution:

```
let rec twistk x k =
    leqk x 1
    (fun b ->
     if b then k 0
     else subk x 1
         (fun r -> twistk r
                   (fun t -> subk x t k)))
```

Workspace

Problem 6. (20 points)

We can describe the Abstract Syntax Trees for an abbreviated portion of PicoML expressions by the following data type:

```
type exp =
   | VarExp of string                    (* variables *)
   | IfExp of exp * exp * exp            (* if exp1 then exp2 else exp3 *)
   | AppExp of exp * exp                 (* exp1 exp2 *)
   | FunExp of string * exp              (* fun x -> exp1 *)
```

Write a function `occurs: string -> exp -> bool` that returns `true` if the string occurs anywhere in the `exp` data structure.

```
# let rec occurs x e = . . .
val occurs : string -> exp -> bool = <fun>
# occurs "a" (IfExp ((VarExp "b"), AppExp(VarExp "a", VarExp "c"), FunExp("a", VarExp "b")));;
- : bool = true
```

Solution:

```
let rec occurs x e =
    match e with VarExp y -> x = y
      | IfExp (b, c, d) -> occurs x b || occurs x c || occurs x d
      | AppExp (f, a) ->  occurs x f || occurs x a
      | FunExp (y, b) -> (x = y) ||  occurs x b
```

Bonus Problem 7. (10 points)

(a) (4 points (bonus)) Create a type of `ifb_list` that can contains any mix of integers, floats and booleans. Your data type should exactly model sequences, possibly empty, of `int`, `float` and `bool`. You may not use the existing type of lists in OCaml in your type.

Solution:

```
type ifb_list =
  | Nil
  | Int_cons   of int * ifb_list
  | Float_cons of float * ifb_list
  | Bool_cons  of bool * ifb_list;;
```

(b) (2 points (bonus)) Represent the mixed list [`true`; 5; 3.4; 6]

Solution:

```
Bool_cons(true,Int_cons(5,Float_cons(3.4,Int_cons(6,Nil))))
```

(c) (4 points (bonus)) Write a function `val shift : ifb_list -> int list * float list * bool list = <fun>` that puts the elements in order into the single typed lists of their own type.

Solution:

```
let rec shift mlist =
    match mlist with Nil -> ([],[],[])
     | Int_cons (n,rest) ->
       let (il,fl,bl) = shift rest in ((n::il),fl,bl)
     | Float_cons (f,rest) ->
       let (il,fl,bl) = shift rest in (il,(f::fl),bl)
     | Bool_cons (b,rest) ->
       let (il,fl,bl) = shift rest in (il,fl,(b::bl))
```

Workspace

Scratch Space

Workspace

A Eval

$\mathsf{Eval}(\texttt{c},\ \rho) = c$ if c is a constant
$\mathsf{Eval}(\texttt{v},\ \rho) = \rho(v)$ if v is a variable
$\mathsf{Eval}(e_1 \oplus e_2,\ \rho) = (\mathsf{Eval}(e_1,\ \rho)) \oplus (\mathsf{Eval}(e_2,\ \rho))$ $\oplus$ a primitive operation
$\mathsf{Eval}(\texttt{fun}\ (x_1, \ldots, x_n)\ \texttt{->}\ body,\ \rho) = \langle (x_1, \ldots, x_n) \to body,\ \rho \rangle$
$\mathsf{Eval}(\texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2,\ \rho) = \mathsf{Eval}(e_2,\ \{x \mapsto \mathsf{Eval}(e_1,\ \rho)\} + \rho)$
$\mathsf{Eval}(f\ e,\ \rho) = \mathsf{Eval}(\mathsf{app}(\mathsf{Eval}(f,\ \rho),\ \mathsf{Eval}(e,\ \rho)),\ \rho)$
$\mathsf{Eval}(\mathsf{app}(\langle (x_1, \ldots, x_n) \to body,\ \rho_1 \rangle,\ (e_1, \ldots, e_n)),\ \rho_2) = \mathsf{Eval}(body,\ \{x_1 \mapsto e_1, \ldots, x_n \mapsto e_n\} + \rho_1)$