

CS421 Spring 2007 Midterm

Friday, February 9, 2007

Name:	
NetID:	

- You have **50 minutes** to complete this exam.
- This is a **closed-book** exam.. You are allowed one 3inch by 5 inch card of notes prepared by yourself. This card is **not to be shared**. All other materials, besides pens, pencils and erasers, are to be away.
- Do not share anything with other students. Do not talk to other students. Do not look at another student's 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 8 pages to the exam. Please verify that you have all 8 pages.
- Please write your name and NetID in the spaces above, and also at the top of every page.

Problems	Possible Points	Points Earned
1	5	
2	9	
3	15	
4	6	
5	10	
6	10	
7	10	
8	10	
Pre Total	75	
Extra Credit 9	7	
Post Total		

CS 421 Midterm 1

Name: _____

1. (5 pts total) Suppose that the following code is input into OCaml:

```
let a = 5;;  
let alist x = [ x ; a ];;  
let a = 7;;  
let b = alist 3;;  
let a = "Bye";;  
let c = alist "Good";;
```

For each of the following, circle every answer that is right.

a. (2 pts) **b** will have a value of

1) [3; 5]

2) [3; 7]

b. (3 pts) After the declaration of **b**,

1) The declaration **let a = "Bye";;** will cause a type error.

2) The declaration **let c = alist "Good";;** will cause a type error.

3) The identifier **c** will have the value ["Good"; "Bye"].

2. (9 pts total)

- a. (3 pts) Write an OCaml function **f:int -> int -> int** that squares the difference of the two arguments and adds the result to the second. Pay attention to the type given.

Solution: `let f x y = let diff = x - y in (diff * diff) + y;;`

What is the result of each of the following applications:

- b. (2 pts) `f 7 4;;`

Solution: 13

- c. (2 pts) `f (4,2);;`

Solution: A type error; **f** needs an int as its first argument, but it is being applied to a pair of ints.

- d. (2 pts) `f 3;;`

Solution: a function of type **:int -> int** that when applied to an argument *y*, squares $(3 - y)$ and adds it to *y*.

CS 421 Midterm 1

Name: _____

3. (15 pts total) Describe the environment that results from the following set of declarations have been made in Ocaml:

```

let a = 5;;
let apair x = (x , a);;
let a = 7;;
let f y = apair (y + a);;
let p = f 2;;

```

You should assume the environment is empty before the declarations are made. Your answer should be a precise mathematical answer, with a precise description of values involved in the environment.

Solution:

```

{a → 7,
 apair → <x → (x, a), {a → 5}>,
 f → <y → apair (y + a), {a → 7, apair → <x → (x, a), {a → 5}>, }>,
 p → (9, 5)}

```

4. (6 pts total) What is printed for each of the following pieces of code? You may omit the final type information.

a. (2pts)

```
let f = fun x -> fun y -> print_string "s"  
in let f1 = f (print_string "t")  
in f1 (print_string "u");;
```

Solution: tus

b. (2pts)

```
let f = fun x -> (let b = print_string "s" in fun y -> print_string "t")  
in let f1 = f (print_string "u")  
in f1 (print_string "v");;
```

Solution: uvst

c. (2pts)

```
let f = let b = print_string "s" in (fun x -> fun y -> print_string "t")  
in let f1 = f (print_string "u")  
in f1 (print_string "v");;
```

Solution: suvt

CS 421 Midterm 1

Name: _____

5. (10 pts) Write a tail recursive function

loop_for : int -> int -> (int -> 'a -> 'a) -> 'a -> 'a

with the following behavior: The first argument is to act as a counter, and is to be incremented by 1 on each successive call. The second argument is the upper bound for the iteration. If **n** is strictly less than **m**, then **loop_for m n f x** should return **x**. Otherwise, **loop_for m n f x** should iteratively apply **f** to the index **m** and the accumulated value **x**, and then repeat with an incremented counter and new accumulated value. (**loop_for** is a functional version of a for-loop.) The expression **loop_for 1 n (fun i -> fun a -> i * a) 1** should calculate the factorial of **n** if **n** ≥ 0 .

Solution: let rec loop_for m n f x =
 if n < m then x else loop_for (m+1) n f (f m x)

6. (10 pts) Write a function **cut_off** : int -> int list -> int list such that **cut_off n lst** returns the list of all elements in **lst** that are less than or equal to **n**. The elements should appear in the same order in the returned list as they did in the input list.

Solution: let rec cut_off n lst =
 match lst with [] => []
 | x::xs -> if x <= n then x::cut_off n xs else cut_off n xs

7. (10 pts) Suppose **lst** is a list of pairs of integers (i.e. **(int * int) list**). Give values for **mult_sum_base** and **mult_sum_step**, so that

List.fold_right mult_sum_step lst mult_sum_base

would return the sum of the product of the pairs in the list. That is, for **lst =**

[(a₁,b₁);(a₂,b₂);...; (a_n,b_n)] it computes **((a₁ * b₁) + (a₂ * b₂) + ... + (a_n * b_n))**

If the list is empty, it should return **0**. The type of **List.fold_right** is

List.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b

- a. (1 pt) What is the type of **mult_sum_base**:

Solution: int

- b. (2 pt) What is **mult_sum_base**:

Solution: 0

- c. (3 pts) What is the type of **mult_sum_step**:

Solution: (int * int) -> int -> int

- d. (4 pts) What is **mult_sum_step**:

Solution: fun (x, y) -> fun r -> (x * y) + r

CS 421 Midterm 1

Name: _____

8. (8 pts)

- a. (3 pts) Write an OCaml variant type **signal** to represent the colors of a traffic light.

Solution: type signal = Green | Yellow | Red

- b. (7 pts) Write a function **change : signal -> signal** that will give the next color a traffic light will show if it is currently showing the given color

Solution: let change sig = match sig with
 Green -> Yellow
 | Yellow -> Red
 | Red -> Green

9. Extra Credit (7 pts total): Write a function **has_duplicates : 'a list -> bool** that returns whether an element occurs at least twice in a list. You may write auxiliary functions, and you may use the library functions List.fold_left, List.fold_right and List.map. You may not use other library functions (including @).

Solution:

```
let mem x l = List.fold_left (fun b y -> b orelse (x = y)) l false
let rec has_duplicates l =
  match l with [] = false
  | (x::xs) = mem x xs orelse has_duplicates xs
```