CMSC330 Spring 2016 Final Exam
Solution

Name: _____

Discussion Time:	**10am**	**11am**	**12pm**	**1pm**	**2pm**	**3pm**
TA Name (Circle):	**Adam**	**Anshul**	**Austin**	**Ayman**	**Damien**	
	Daniel	**Jason**	**Michael**	**Patrick**	**William**	

Instructions

- The exam has 15 pages; make sure you have them all.
- Do not start this test until you are told to do so!
- You have 120 minutes to take this midterm.
- This exam has a total of 130 points, so allocate 55 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- Answer essay questions concisely in 2-3 sentences. Longer answers are not needed.
- For partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

	Problem	Score
1	PL Concepts	/16
2	Lambda Calculus	/8
3	OCaml Types	/8
4	OCaml Execution	/18
5	OCaml Programming	/12
6	Ruby Programming	/14
7	Prolog Execution	/12
8	Prolog Programming	/12
9	Regexps, FAs, CFGs	/18
10	Security	/12
	TOTAL	/130

1. PL Concepts (16 pts)

1. (2 pts) Which garbage collection algorithm can successfully clean up a cyclical linked list data structure?

A. Cyclic data structures cannot be garbage collected.
B. Reference Counting
C. Mark-and-Sweep
D. Any garbage collection algorithm can clean it.

2. (2 pts) T/**F** : Garbage Collected languages are immune to memory leaks.

3. (2 pts) Which of the following best describes the *declarative* programming paradigm.

A. Programs are described in terms of the desired results of computation.
B. Programs are described in terms of the control flows of computation.
C. Programs use expressions as opposed to statements.
D. Programs use statements as opposed to expressions.

4. (4 pts)Give the output of the following C code, first evaluated as Call-by-Value, then as Call-By-Reference

```
#include <stdio.h>
#define ARR_LEN 5

int arr[ARR_LEN] = {1,1,1,1,1};

void fun(int x,  int y) {
    x += 1;
    arr[x] = 0;
    y += x;
    arr[y] = y;
}

int main(int argc, char **argv) {
    int a = 0, b = 1;
    fun(a, b);
    fun(a, b);

    int i;
    for (i = 0; i < ARR_LEN; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

1) Call-by-Value: **1 0 2 1 1**

Call-by-Reference: **1 0 0 1 4**

5. (2 pts) Which of the following best describes the *imperative* programming paradigm.
   A. Programs are described in terms of the desired results of computation.
   B. Programs are described in terms of the control flows of computation.
   C. Programs use expressions as opposed to statements.
   D. Programs use statements as opposed to expressions.

6. (2 pts) What are the necessary components of a closure? (Circle all that apply)
   A. Garbage collection
   B. A mapping of free variables to bindings and scopes
   C. A function to call.
   D. Static scoping

7. (2 pts) Consider the languages C and OCaml. Give an example of a functional paradigm feature that OCaml has which C does not, and explain why it is useful.

   Possibilities:
   a) Currying
   b) First class functions with closures
   c) Strict typing
   d) No mutable states
   e) Implicit module system
   f) NOT just first class functions or higher order functions, those work in C too. Need to make mention of closures being an improvement.

2. Lambda Calculus (8 pts)

1. (2 pts) Tomatoes want to show that the circle is the ultimate shape. Circle the bound variables in the following lambda expression.

   λ x . λ y . (λ z . x z) z y

   Solution: λ x . λ y . (λ z . **x z**) z **y**

2. (6 pts total) The brussel sprouts are multiplying! First there was one, but now there are two more! Given the following Church numeral encodings, how many brussel sprouts do you have to eat?

   Addition: λm. λn. λf. λx. m f (n f x)
   The Number 1: λf. λx. f x
   The Number 2: λf. λx. f (f x)

The Number 3: λf. λx. f (f (f x))

Show that 1 + 2 = 3 (Hint: leave the numbers as numbers for as long as possible)

Start:	**Addition 1 2**
Expand:	**(λm. λn. λf. λx. m f (n f x)) 1 2**
Beta reduction:	**(λn. λf. λx. 1 f (n f x)) 2**
Beta reduction:	**λf. λx. 1 f (2 f x)**
Expand:	**λf. λx. (λf. λx. f x) f (2 f x)**
Beta reduction:	**λf. λx. (λx. f x) (2 f x)**
Beta reduction:	**λf. λx. f (2 f x)**
Expand:	**λf. λx. f ((λf. λx. f (f x)) f x)**
Beta reduction:	**λf. λx. f ((λx. f (f x)) x)**
Beta reduction:	**λf. λx. f (f (f x))**
Compress:	**3**

3. OCaml Types (8 pts)

1. What is the type of the following OCaml expression?

```
[("root",[1]); ("ahh",[2]); ("bay",[3]); ("gah",[4])]
```

(string * int list) list

2. What is the type of the following OCaml function?

```
let f huh nee du = (huh (nee du)) > du
```

('a -> 'b) -> ('b -> 'a) -> `b -> bool

3. Write an OCaml expression with type `(`a -> int) -> `a list -> int`. You are not allowed to use type annotations.

```
fun f h::t -> (f h) + 1;;
```

4. Write an OCaml expression with type `` `a * `b -> `c * `d -> (`a * `c) list ``. You are not allowed to use type annotations.

```
fun (a,b) (c,d) -> [(a,c)];;
```

4.OCaml Execution (18 pts)

```
let rec map f l = match l with
     [] -> []
     | h::t -> let r = f h in r :: map f t
;;

let rec fold f a l = match l with
    [] -> a
   |h::t -> fold f (f a h) t
;;
```

What is the output of the following OCaml expressions? If an error will occur, briefly describe what the problem is.

1. (3 pts)

```
let transform lst f =
    let lst' = map f lst in
    fold (fun a e -> a + " " + e) "" lst'
;;
transform [3;9;6] (fun e -> string_of_int e)
```

Error: '+' operator does not work with strings. '^' is used for string concatenation.

2. (4 pts)

```
let rec exp x y = match x with
  | [] -> []
  | h::t -> if y then h::(exp t (not y)) else exp t (not y)
exp [1;2;3;4;5;6;7;8] false
```

[2; 4; 6; 8]

3. (4 pts)

```
let f a b c d=
  let t = a b c in
  d t a;;
f (fun x y -> x*y) 2 3 (fun x y -> y x x)
```

36

4. (3 pts)

```
let mystery list =
    let rec aux acc = function
      | [] -> acc
      | h::t -> aux (h::acc) t in
    aux [] list;;
 mystery ["a" ; "b" ; "c"];;
```

[‘c’; ‘b’; ‘a’]

5. (4 pts)

```
let x = ref 5;;
fold (fun a h -> x := (!x) + h; a + (!x)) 0 [1; 2; 3];;
```

25

5.OCaml Programming (12 pts)

A. Given the definition of integer Binary Search Tree bst

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

1. (4 pts) Implement a function insert of type bst -> int -> bst, which will take a bst, insert a number to the correct position in a bst and then return the tree. You can assume that all the int values in the bst are distinct. In a binary search tree, we examine the root and recursively insert the new node to the left subtree if its key is less than that of the root, or the right subtree if its key is greater than or equal to the root.

let rec insert tree n =

```
let rec insert tree n = match tree with
          | Empty -> Node(Empty, n, Empty)
          | Node(left, num, right) -> if (n < num) then
            Node((insert left n), num, right)
     else
          Node(left, num, insert right n)
;;
```

2. (4 pts) Write a map function of type "(int->int) -> bst -> bst" for bst. map produces tree of same shape, every element is obtained by applying f to each node of tree.

For example:
map (fun x -> x * 3) bst will return a bst, whose node values are 3 times of those of bst.

let rec map f bst =

```
let rec map f tree = match tree with
          | Empty -> Empty
          | Node(left, num, right) ->
                Node(map f left, f num, map f right)
;;
```

B.(4 pts) Write a function compress : 'a list -> 'a list, which eliminates consecutive duplicates of list elements.

For example: compress ["a";"a";"a";"a";"b";"c";"c";"a";"a";"d";"e";"e";"e";"e"]
returns ["a"; "b"; "c"; "a"; "d"; "e"]

```
let rec compress = function
        | a::(b::_ as t) -> if a = b then
                compress t
          else
                a::(compress t)
        | a -> a
;;
```

6. Ruby Programming (14 pts)

Farmer Saurav wants to update his tools to the 21st century. In doing so, he needs help writing a ruby script that reads in location data about his crops and performs useful operations on it.

He needs you to implement the following functions:
A. (5 pts) initialize(file) - takes in a string with the filename of the input data and stores that file data into an appropriate data structure. In Ruby IO.foreach opens a file calls the given block for each line it reads and closes the file afterwards. You don't have to worry about closing the file. For example: IO.foreach("file") { |line| puts line }

B. (5 pts)get_vicinity(x, y, size) - returns array of crop ids within the square that has bottom left corner (*x*,*y*) and side length *size* (i.e. top-right corner is (x + size, y + size). This should include crop ids along the borders of the square.

C. (4 pts) search(string) - returns a hash mapping crop ids whose notes ***contain*** the input string to their notes. See the example input / output for more clarification.

Farmer Saurav has been very careful, so you can assume the input file contains only correctly formatted input.

== Example input file: ===
This is a comment
Format is <crop id>;<x coordinate>;<y coordinate>;<farmer notes>
0;12.3;3.2;corn
22;4.2;1.1;cabbage patch
====

Example input/output:

Command	**Output**
`crops = CropManager.new("input.txt")`	
`puts crops.get_vicinity(12, 3, 1).to_s`	`[0]`
`puts crops.get_vicinity(4, 1, 1).to_s`	`[22]`
`puts crops.get_vicinity(4, 1, 9).to_s`	`[0,22]`
`puts crops.get_vicinity(1, 0, 3).to_s`	`[]`
`puts crops.search("cabbage").to_s`	`{22=>"cabbage patch"}`
`puts crops.search("c").to_s`	`{0=>"corn", 22=>"cabbage patch"}`
`puts crops.search("the answer").to_s`	`{}`

The crop id will always be a unique positive integer.
The x and y coordinates will always be positive floating point numbers.
The farmer notes will never contain semicolons.

```
class CropManager
def initialize(file)
      @vals = {}

      IO.foreach(file) {|line|
            parts = line.split(“;”)
            @vals[parts[0].to_i] = [parts[1].to_f, parts[2].to_f,
      parts[3]]
      }
end
```

def get_vicinity(x, y, width)

```
res = []

      @vals.each{|k, v|
            if v[0] >= x && v[0] <= x + width && v[1] >= y && v[1] <= y
+ width
                  res.push(k)
            end
      }

      return res
end
```

def search(string)

```
res = {}

      @vals.each{|k, v|
            res[k] = v[2] if v[2] =~ /#{string}/
      }

      return res
```

end

7. Prolog Execution (12 pts)

```
veggie(arugula).
veggie(broccoli).
meat(steak).
meat(shrimp).

vegetarian(jack).
vegetarian(jim).
meatEater(amy).
meatEater(paul).

eats(X,Y) :-
      vegetarian(X),
      veggie(Y).

eats(X,Y) :-
      meatEater(X), !,
      meat(Y).

eats(X,Y) :-
      not(meatEater(X)),
      not(vegetarian(X)),
      meat(Y).

eats(X,Y) :-
      not(meatEater(X)),
      not(vegetarian(X)),
      veggie(Y).
```

For each query, list the substitution X that Prolog produces that makes the query true. If there is more than one answer, list them all, separated by semi-colons. If there is no answer, write false. Answers must be listed in the correct order to receive full credit.

A. eats(X,steak).

X = amy

B. eats(X,arugula).

X = jack, X = jim.

C. eats(maurie,X).

X = steak, X = shrimp, X = arugula, X = brocolli.

D. eats(jim,X).

X = arugala, X = brocoli.

8. Prolog Programming (12 pts)

Note: for the following questions, you may only use the library functions member and append.
A. (6 pts) Create a function remdup that takes in a list of elements and returns a list with all the duplicates removed **left to right**.

Example:

```
?- remdup([a, a, b, c, d, a, e, d, b, c, f], R).
R = [a, b, c, d, e, f].
remduphelp([],_,[]).
remduphelp([H|T], S, R) :- member(H, S), remduphelp(T, S, R), !.
remduphelp([H|T], S, [H|R]) :- remduphelp(T, [H|S], R).
remdup(L, R) :- remduphelp(L, [], R).
```

B. (6 pts) Create a function inds that takes in a list of indices (zero-based) and a list of elements, and returns a list of the elements that correspond to the indices. You may assume all the indices are in-bounds.

Example:

```
?- inds([1, 3, 5], [a, b, c, d, e, f, g], R).
R = [b, d, f].
```

```
indshelp(_, [], _, []).
indshelp(X, [H|T], I, [H|R]) :-
      member(I, X), I1 is I + 1, indshelp(X, T, I1, R).
indshelp(X, [_|T], I, R) :- I1 is I + 1, indshelp(X, T, I1, R).
inds(X, Y, R) :- indshelp(X, Y, 0, R), !.
```

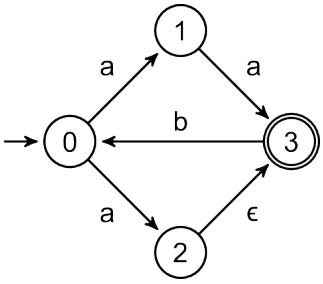
9. Regexps, Finite Automata, and CFGs (18 pts)

1. (3 pts) Write a regular expression which recognizes set of strings over alphabet {a,b} that do not end in aaa

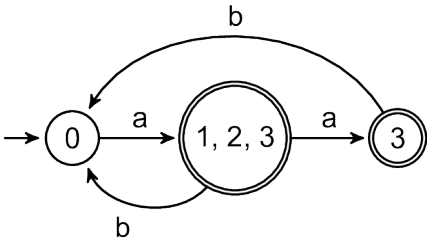
```
/^((a|(aa))|((a|b)*(ba?a?)))?$/
```

2. (3 pts) Describe the strings that the following NFA accepts.

`((aa|a)b)*(aa|a)` OR `(aa|a)((aa|a)b)*`



3. (5 pts) Convert the above NFA to an equivalent DFA.



4. (3 pts) Write a context-free grammar which accepts the following language.

$a^n b^n a^m b^m$ where $n, m \geq 0$

S -> TT
T -> aTb | e

5. (4 pts) The following grammar cannot be parsed with a recursive descent parser. Rewrite it to fix any problems.

S -> raA | rD
A -> Ab | iD
D -> tD | t

S -> rX
X -> aM | D
M -> iDbA
A -> bA | e
D -> tE
E -> tE | e

10. Software Security (12 pts)

A. (2 points) I have a buggy web application, called SellRE, where users can post text to a webpage. If this webpage will execute code contained in posts, to what exploit is the code vulnerable?

A. Command Injection
B. Buffer Overflow
C. Cross-Site Scripting
D. SQL Injection

B. (2 points) There's a zucchini hiding among the cucumbers! Using the concept of minimal privilege, by doing which of the following can you prevent an untrusted user like that zucchini from doing nasty things to your application? Choose all that apply:

A. Making a non-root process handle untrusted input.
B. Preventing non-account holders from being tracked.
C. Compartmentalizing information without securing user permissions.
D. **Reducing privileges as much as possible**.
E. Assuming that all user input is valid.

C. Good King Henry has written an FTP server much like the one in Project 6. However, he knows that there are bugs in it, and he wants to squash all of them. Take a gander at this code for a function called `remove`, which removes a file from the server. The file can **only be removed from the public directory or from the logged in user's private directory** (each user has a `user.txt` file in his or her private directory). The input `arg` should be the name of the file to be deleted:

```
def remove(arg, socket)
  unless logged_in socket
    my_send(socket, "You're not logged in yet")
    return
  end
  if arg =~ /^public\// then
    full_path = File.join(Dir.pwd, "users", arg)
  else
    full_path = File.join(@s2a[socket].directory, arg)
  end
  system("rm #{full_path}")
  my_send(socket, "Removed file: full_path")
end
```

a) (2 points) Mark in the above code where the vulnerability/vulnerabilities are, and label them.

Both command injection and path traversal vulnerabilities existed in the code.

b) (3 points) Give an input or inputs that the client could use to exploit the vulnerability or vulnerabilities you identified in the above code, assuming that there exist accounts for users rughoula and kappage, and that the client is logged in as rughoula. Also assume each user has a user.txt file in his or her private directory.

Code injection: `remove user.txt; rm -rf /`
Path traversal: `remove public/../kappage/user.txt`

c) (3 points) Show how to fix the code to eliminate the problem(s). If you can't remember particular Ruby syntax or commands, explain as best you can.

Blacklisting paths containing `/../` or semicolons, or removing `/../` or semicolons from paths.