CMSC330 Fall 2021 Final Exam
Sections 010x and 020x
Solutions

Name (PRINT YOUR NAME as it appears on gradescope):

_____

Instructions

- The exam has 15 pages (front and back); 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 exam.
- 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	/8
2	Lambda Calculus	/10
3	OCaml	/20
4	Ruby	/15
5	Rust	/10
6	Regexps, FAs, CFGs	/14
7	Parsing	/10
8	Operational Semantics	/6
9	Security	/7
	TOTAL	/100

1. PL Concepts [8 pts]

Circle your answers. Each T/F question is 1 point.

1) T / F OCaml and Ruby use garbage collection.

2) T / F Let statements declare immutable variables by default in OCaml and Rust.

3) T / F When using a language with a call-by-value evaluation strategy, arguments are reduced to a value before being passed onto a function .

4) T / F Tail recursive functions can be optimized by a compiler to not cause stack overflows .

5) T / F If the program terminates, CALL BY VALUE and CALL BY NAME will always generate the same value.

6) T / F OCaml is not statically-typed because you don't have to explicitly declare types.

7) T / F Reference counting and mark and sweep are both techniques used by garbage collection algorithms.

8) T / F All NFAs are DFAs, but not all DFAs are NFAs.

2. Lambda Calculus [10 pts]

A. Make the parentheses explicit.

1) [2 pts] `(λx. x z) λy. w λw. w y z x`

```
(λx. (x z)) (λy. (w (λw. (((w y) z) x))))
```

2) [2 pts] `λx. x y λx. y x`

```
(λx. ((x y) (λx. (y x))))
```

B. Reduce the following lambda expressions using **Call By Name** and **Call By Value** strategies. Make appropriate use of parentheses and alpha reductions to get maximal partial credit.
`(λx. x y) ((λy. a) (λx. y))`

[3 pts] **Call By Name**

```
(λx. x y) ((λy. a) (λx. y))
 = ((λy. a) (λx. y)) y
 = a y
```

[3 pts] **Call By Value**

```
(λx. x y) ((λy. a) (λx. y))
 = (λx. x y) a
 = a y
```

3. OCaml [20 pts]

A. Write the types of the following OCaml expressions. If the expression doesn't type check, just write "type error" with no explanation required.

1) [3 pts] `fun x -> fun y -> [x = (y + 1)]`

```
int -> int -> bool list
```

2) [3 pts] `fun x y z -> match (x = y z) with true -> y z | false -> y x`

```
'a -> ('a -> 'a) -> 'a -> 'a
```

B. [4 pts] What would you put in place for the blank such that the following code returns **44**.

```
let f = fun x y z ->
    List.fold_left(fun acc ele -> acc + (ele + (y z)))  0 x
    in f [1; 1; 1; 1] _________________ 4
```

```
(fun x -> x + 6)
```

For the questions C and D, you may **not** use the **List module** or **@**, but you may use the helper functions given below. You may also write your own helper functions.

Helper Functions:

```
let rec map f xs = match xs with
| [] -> []
| x::xt -> (f x)::(map f xt)

let rec foldl f a xs = match xs with
| [] -> a
| x::xt -> foldl f (f a x) xt

let rec foldr f xs a = match xs with
| [] -> a
| x::xt -> f x (foldr f xt a)
```

C. [5 pts] Write a function **make_palindrome** of type **'a list -> 'a list** which makes the input list into a palindrome.

Examples:

```
make_palindrome [] = []
make_palindrome [1; 2; 3] = [1; 2; 3; 3; 2; 1]
make_palindrome [“c”; “m”; “s”; “c”; “c”; “s”; “m”; “c”]
```

```
let rec make_palindrome lst =
  let rev = foldl (fun a x -> x::a) [] lst in
  foldr (fun x a -> x::a) lst rev
```

D. [5 pts] Write a function **lod** of type **int list -> int list -> int list** that creates a new list that is the result of subtracting each item of the second list from each item at the same index in the first lst. Assume two lists have the same size.

Examples:

```
lod [] [] = []
lod [1; 2; 3] [3; 2; 1] = [-2; 0; 2] (*[(1-3);(2-2);(3-1)] *)
```

```
let rec lod lst1 lst2 =
  match lst1, lst2 with
  | [],[] -> []
  | h1::t1,h2::t2 -> (h1-h2)::(lod t1 t2)
```

4. Ruby [15 pts]

In 2019, the University of Maryland awarded a $100 million contract to Workday to upgrade Testudo to a new cloud-based system. You were hired by Workday to develop a part of the course registration system because of your experience in registering for classes and your Ruby skills.

The goal of the course registration system is to allow students to add/drop classes and see all the courses they've registered for. You will be given a file called **courses.txt** which contains information about every course offered by the university. Each line in **courses.txt** will have the format:
`Course,Seats`

The course name will consist of 4 uppercase letters followed by 3 digits and the total number of seats will be a number with one or more digits. **All invalid lines should be ignored**. For example, the following line is valid: `CMSC330,30`

Each student will be represented as a string. We will simply denote this as `id`.
You will have to implement four functions, described below:

- [3 pts] `initialize(path)` - Reads the file and parses the contents. Store the contents in any data structure you like, as long as these other functions work as described below.
- [4 pts] `add(id, course)` - Registers a student to the given course and returns `true` if there is an open seat and the student has not already registered for the course. Otherwise, return `false`. Note that the total number of seats given in **courses.txt** is not the same as the total number of open seats. You will have to keep track of open seats.
- [4 pts] `drop(id, course)` - Drops a student from the given course if the student has registered for it and returns `true`. In all other cases, return `false`.
- [4 pts] `get_courses(id)` - Returns an array containing all the courses a student has registered for. if there's no such student/no registration, return an empty array.

The class definition is given on the following page for you to fill in. Feel free to add any class or instance variables you feel are necessary. Here is an example interaction with the class:

```
r = CourseRegistration.new('courses.txt')
r.add('1001', 'CMSC351')
=> true
r.add('1001', 'CMSC330')
=> true
r.add('1002', 'CMSC330')
=> true
r.add('1010', 'CMSC330')
=> false    # No more open seats! :(
```

courses.txt:

```
CMSC330,2
INVALID,0
CMSC351,3
ENGL393,2
STAT400,2
```

r.drop('1002', 'CMSC330') => true r.get_courses('1001') => ["CMSC330", "CMSC351"]	

```
class CourseRegistration
  def initialize(path)
      @courses = {}

    File.open(path).each do |line|
      if line =~ /^([A-Z]{4}\d{3}),(\d+)$/
        @courses[$1] = [$2.to_i, []]
      end

    end
  end

  def add(id, course)
     if @courses[course] && @courses[course][1].count < @courses[course][0] &&
     !@courses[course][1].include?(id)
       @courses[course][2] << id
       true
     end
  end false

  def drop(id, course)
    if @courses[course]
      @courses[course][2].delete(id)
      true
    end
    false
  end

  def get_courses(id)
        list = []
        @courses.each do |k,v|
          if v[1].include?(id)
            list << k
          end
        end
  end   list
end
```

5. Rust [10 pts]

A. [3 pts] Does the following program compile? If so, write out the output of the program execution. Otherwise, point out the line that causes the error and explain why the program doesn't compile.

```
1  fn main () {
2    let mut a = 3;
3    let b = &mut a;
4    let &mut c = b;
5    *b = 5;
6    println! ("{}", a + c);
7  }
```

Compiles and prints 8

B. [3 pts] Who is the owner of the string “330 rocks!” when the execution stops at **HERE**?

```
1  fn main () {
2    let a = String::from("330 rocks!");
3    let b = a;
4    let c = &b;
5    let d = &c;
6    let e = d;
7    let f = &*c;
8    let g = e;
9    // HERE
10 }
```

b

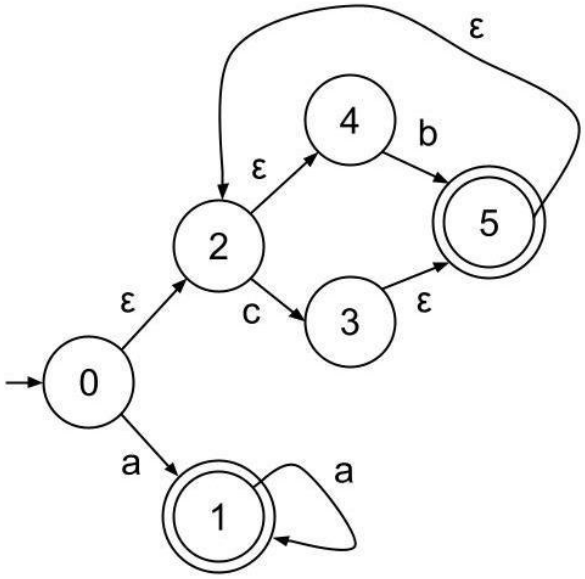
C. [4 pts] The function **`add_elems`** sums up the elements of an integer array. Find and fix the errors in the function. Find three - there may be more, any three are fine.

```
1  fn add_elems(arr : &[i32]) -> i32 {
2    let sum = 0;
3    for i in arr.iter() {
4      sum += arr[i];
5    }
6    sum;
7  }
```

1. sum must be mutable
2. Should be adding i instead of arr[i]
3. Line 6 should not have a semicolon

6. Regex, FAs, CFGS [14 pts]

A. [6 pts] Use the subset algorithm to convert the above NFA to a DFA.
Show all steps and draw the final DFA.



Final DFA after subset construction should have the following formal definition:

Σ: {a, b, c}
Q: {{0,2,4}, {1}, {5,2,4}, {3,5,2,4}}
q0: {0,2,4}
F: {{1}, {5,2,4}, {3,5,2,4}}
d: {({0,2,4}, a, {1}), ({0,2,4}, b, {5,2,4}),
({0,2,4}, c, {3,5,2,4}), ({1}, a, {1}), ({5,2,4}, b, {5,2,4}),
({5,2,4}, c, {3,5,2,4}), ({3,5,2,4}, b, {5,2,4}),
({3,5,2,4}, c, {3,5,2,4})}

B.[4 pts] Write a regex to describe the language of the above NFA:

```
(a+)|((b|c)+)
```

C. [4 pts] Write a context free grammar that generates the following language:
$a^x b^y c^z$ where $z = x + y$, $x \geq 0$ and $y \geq 0$

```
S -> aSC | T
T -> bTc | ε
```

7. Parsing [10 pts]

A. [3 pts] List the first sets for the following grammar:
You may use e or ε for epsilon. **Do not use E as that could imply a non-terminal.**

S → aB | Rw
B → dR | n
R → c | ε

```
First(S) = {a, c, w}
First(B) = {d, n}
First(R) = {c, ε}
```

B. [3 pts] Fix the following grammar so that it can be parsed by a recursive descent parser

S → SAB| AB
A → Aa | a
B → Bb | b

```
S -> ABS | AB
A -> aA | a
B -> bB | b
```

C. [4 pts] The following is the CFG for an even length palindrome over the alphabet {a,b}

S → aSa | bSb| ε

Can you write a recursive descent parser for this grammar? If so, please write the **parse_S** function using either of the two implementations of `lookahead` or `match_tok` covered in the class. Assume the token list type of `char list`. For example: `['a'; 'b'; 'b'; 'a']`.
If a recursive descent parser cannot parse this grammar, explain why.

```
CFG cannot be parsed by recursive descent because there is no unique
first set for ε case, making the first and second halves of
palindrome indistinguishable under recursive descent.
```

8. Operational Semantics [6 pts]

$$\frac{}{A;\ \texttt{false} \Rightarrow \texttt{false}}\ (1) \qquad \frac{}{A;\ \texttt{true} \Rightarrow \texttt{true}}\ (2) \qquad \frac{}{A;\ n \Rightarrow n}\ (3) \qquad \frac{A(x) = v}{A;\ x \Rightarrow v}\ (4)$$

$$\frac{A;\ e_1 \Rightarrow n_1 \qquad A;\ e_2 \Rightarrow n_2 \qquad v \text{ is } n_1 > n_2}{A;\ e_1 > e_2 \Rightarrow v}\ (5) \qquad \frac{A;\ e_1 \Rightarrow \texttt{true} \qquad A;\ e_2 \Rightarrow v}{A;\ \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \Rightarrow v}\ (6)$$

$$\frac{A;\ e_1 \Rightarrow n_1 \qquad A;\ e_2 \Rightarrow n_2 \qquad v \text{ is } n_1 + n_2}{A;\ e_1 + e_2 \Rightarrow v}\ (7) \qquad \frac{A;\ e_1 \Rightarrow \texttt{false} \qquad A;\ e_3 \Rightarrow v}{A;\ \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \Rightarrow v}\ (8)$$

$$\frac{A;\ e_1 \Rightarrow v_1 \qquad A, x : v_1;\ e_2 \Rightarrow v_2}{A;\ \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 \Rightarrow v_2}\ (9)$$

Consider the following incomplete (but partially filled!) proof tree using the above semantics:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A;\ 4 \Rightarrow 4}\ (3) \qquad \text{———}\ (\boxed{\text{c}}) \qquad 12 \text{ is } 4+8}{\text{———}\ (\boxed{\text{b}})} \qquad \dfrac{}{A;\ 15 \Rightarrow 15}\ (3) \qquad ?\text{ is }?}{\text{———}\ (\boxed{\text{a}})} \qquad \text{———}\ (\boxed{\text{d}})}{A;\ \texttt{if } 4+8 > 15 \texttt{ then } 16 > 23 \texttt{ else } 42 \Rightarrow \boxed{???}}\ (8)$$

A. [4 pts] What rule numbers correspond to the holes in the tree?
ONLY WRITE THE RULE NUMBERS. DO NOT WRITE MORE THAN ONE DIGIT PER ITEM.

a) 5
b) 7
c) 3
d) 3

B. [2 pts] What is the final answer? (In other words: what value should go in the box ??? in the bottom of the proof tree?)

42

9. Security [7 pts]

A. [3 pts] There is a security vulnerability in the following code. Explain how the attacker can exploit the vulnerability assuming that stack is allocated high to low address.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void print_strings(char *buffer, int len) {
  for (int i=0; i<len; i++) {
    if (buffer[i] != 0) {
      printf("%c", buffer[i]);
    }
  }
  printf("\n");
}

int main(int argc, char **argv) {
  char secret[32];
  char public[32];
  strcpy(secret, "This is a secret");
  strcpy(public, "This is public data");
  int len = atoi(argv[1]);

  char buffer[len+1];
  memcpy(buffer, public, len);
  buffer[len] = 0;
  print_strings(buffer, len);
}
```

Buffer overflow. Large value for argv[1] can read memory outside of what's allowed i.e secret array.

B. [2 pts] SQL injections are best prevented by (select all that apply) :

a) Input Sanitization
b) Prepared Statements
c) Forced Type Checking
d) Dynamic Memory Allocation
e) Randomized Memory Access

C. [2 pts] A stored XSS attack typically can be prevented if the web servers

a) never output data received as input directly without checking it for malicious code
b) use the same origin policy
c) disable javascript
d) use prepared statements