CMSC330 Fall 2019 - Final Exam

SOLUTIONS

First and Last Name (PRINT): ______________________________

9-Digit University ID: ______________________________

Instructions:

- 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 100 points.
- 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.
- **Write your 9-Digit UID at the top of EVERY PAGE.**

1. PL Concepts	/ 8
2. Lambda Calculus	/ 8
3. OCaml	/ 24
4. Ruby	/ 12
5. Rust	/ 10
6. RegExp, FA, CFG	/ 12
7. Parsing	/ 8
8. Operational Semantics	/ 10
9. Security	/ 8
Total	/ 100

Please write and sign the University Honor Code below: **I pledge on my honor that I have not given or received any unauthorized assistance on this examination.**

I solemnly swear that I didn't cheat.

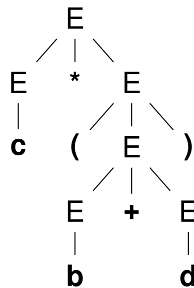
Signature: ______________________________

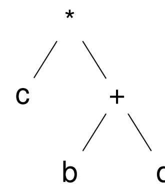
1. [8pts] PL Concepts

1 (5pts) **Circle your answers.** Each T/F question is 1 point.

T	**F**	Ruby has static type checking
T	**F**	Purely functional data structures in the functional programming paradigm are mutable
T	**F**	Two items that have structural equality must also have physical equality
T	F	A key feature of most scripting language is implicit declarations
T	F	In the call-by-name method of evaluation, we delay argument evaluation until the value of an argument evaluation is used

2 (1pts) The two trees shown below are involved in parsing the expression `c*(b+d)`. Identify which is the Abstract Syntax Tree and which is the Parse Tree. You can label AST or PT.





Parse Tree	**Abstract Syntax Tree**

3 (2pts) Order the following from the one that occurs first to the one that occurs last (number them 1 to 3)

Evaluation (Interpreter)	**Token Generation (Lexer)**	**AST Creation (Parser)**
3	**1**	**2**

2. [8pts] Lambda Calculus

1 (4pts) The following OCaml type represents lambda expressions:

```
type expr =
    | App of expr * expr
    | Lam of string * expr
    | Var of string
```

Give the expr value for the following lambda expressions (you may want to draw parentheses first, explicitly). For example, for $\lambda x.x$ you would write `Lam ("x", Var "x")`.

a) $\lambda x.x\ y$

```
Lam("x", App(Var "x", Var "y"))
```

b) $\lambda x.x\ \lambda y.y\ w$

```
Lam("x", App(Var "x", Lam("y", App(Var "y", Var "w"))))
```

2 (4pts) Prove the following using the appropriate λ-calculus encodings (show all steps for full credit):

`or true false ⇒ true`

Some definitions that may be helpful:

or: $\lambda x.\lambda y.((x\ \text{true})\ y)$
true: $\lambda x.\lambda y.x$
false: $\lambda x.\lambda y.y$

Write your proof here:

(or true false)
(λx. λy. ((x true) y) true false)
(λy. ((true true) y) false)
((true true) false)
(((λx. λy. x) true) false)
((λy. true) false)
true

3. [24pts] OCaml

1 (4pts) Give the types of the following snippets of OCaml code. If an error is raised, state the error.

a) `fun f h k -> f (h + k) = 2.0`

```
(int -> float) -> int -> int -> bool
```

b) `let f b c d = (c (d::b))::b`

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

2 (2pts) Write a code snippet in OCaml that has the given type. Do NOT use type annotations.

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

```
fun x y z -> (z x)::[y]
```

3 (2pts) Given the following code

```
let add x y = print_int (x+y); x + y;;
let e1 = lazy (add 10 20);;
let e2 = lazy (add 1 2);;
let foo p e1 e2 = Lazy.force(if p then e1 else e2);;
```

What is the return value of `foo true e1 e2`?

30

4 (2pts) Given the module Foo

```
module type FOO =sig
  val mul : int -> int -> int
end;;
module Foo : FOO =struct
   let add x y = x + y
   let mul x y = x * y
end;;
```

What is the result of each of the following?

a) `Foo.add 3 4`

ERROR (add is not in the signature, and is therefore not visible)

b) `Foo.mul 3 4`

12

5 (6pts) Write a function `is_prime` that returns `true` if the argument provided is a prime number, and `false` if it is not a prime number. You **may NOT** use any imperative features of OCaml.

Remember that a number is prime if it is divisible by exactly 2 distinct positive numbers that are less than or equal to itself (1 is *not* a prime number). You can assume inputs to your function will be positive. You may define any inner helper functions you would like, and you may use the function definitions below. You may use the built-in `mod` function in OCaml. The syntax for this is `a mod b`, which compute the value of `a` modulo `b`. For example `10 mod 3` = `1`.

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

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

```
(* Examples *)
is_prime 1 = false
is_prime 5 = true
is_prime 8 = false
```

Implement the function below:

```
let rec is_prime num =
    if num = 1 then
        false
    else
        let rec helper curr =
            if (curr < num) then
                curr :: (helper (curr + 1))
            else
                []
        in
            fold (fun a h -> a && (num mod h) != 0) true (helper 2)
```

6 (8pts) Write a function in OCaml to give the frequency of characters in a string. The result should contain only unique letters, like a set. The order of the list does not matter. For example:

```
char_freq "hello" = [('h', 1); ('e', 1); ('l', 2); ('o', 1)]
```

For the sake of this part alone, assume that you are given the below function `explode` which converts a `string` into a `char list`. For example:

```
# explode "hello";;
- : char list = ['h'; 'e'; 'l'; 'l'; 'o']
```

You may use any function from the `List` module, including `map`, `fold`, `sort`, or any of the `assoc` functions. A few are shown below to help you:

`List.sort (compare) ['a';'c';'a';'b';'a'] = ['a';'a';'a';'b';'c']`
`List.assoc: 'a -> ('a * 'b) list -> 'b` returns the value associated with key `a` in the list of pairs. Raises Not_found if there is no value associated with `a` in the list.
`List.mem_assoc: 'a -> ('a * 'b) list -> bool` returns true if a binding exists, and false if no binding exists for the given key.
`List.remove_assoc: 'a -> ('a * 'b) list -> ('a * 'b) list` removes the first pair with key `a`, if any.

You can add helper functions, and they can be recursive, but DO NOT use any imperative features of OCaml.

Implement the function below:

```
let char_freq s =
    let f acc x =
        if (mem_assoc x acc) then
            let count = assoc x acc in
                (x, count + 1) :: (remove_assoc x acc)
            else
                (x, 1) :: acc
    in
        fold_left f [] (explode s)
```

4. [12pts] Ruby

Going to office hours for CMSC330 often involves a lot of waiting. We want to find out how much time, in total, students waited in the office hours room this semester. You will be given a file called **waiting.txt** which contains information about each time a student waited in office hours.

If a student goes to office hours more than once, their name will appear in the file multiple times, once for each time they went to office hours. Each line in **waiting.txt** should be of the following format:

```
Lastname, Firstname, Minutes:Seconds
```

Both the first name and last name begin with an uppercase letter, followed by one or more lowercase letters. Each commas should always be followed by a single space. There are no spaces around the colon between the minutes and seconds. ***INVALID LINES SHOULD BE IGNORED***. For example, if Santa Claus waited for 15 minutes and 32 seconds on one day, you would see the line

```
Claus, Santa, 15:32
```

The time is always represented as minutes and seconds, so if a student waits for two hours and 15 minutes, their time will be `135:00`. There should always be a number before the colon, so for example, `:30` is invalid, but `0:30` is valid. For simplicity, leading zeros for the minutes are allowed, but there will always be exactly 2 digits for seconds (`00:30` is allowed, as is `0010:15`).

You will have to implement three functions, described below:

- **`initialize(filename)`** 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.
- **`student_waited_for(student_name)`** Determines how long a particular student waited for. The `student_name` will be in the format “Firstname Lastname” (note here, there is no comma, and the firstname is first, the opposite of in the file). The time should be returned in seconds. To find the total number of seconds, multiply the minutes by 60 and add the seconds. For example, 5:30 = 5 * 60 + 30 = 330 seconds.
- **`total_wait_time()`** Adds up the total time added by all students, and returns the time in seconds. You may assume the `student_waited_for` method was implemented correctly and use it here if you would like.

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:

`w = WaitingTime.new("waiting.txt")`
`w.student_waited_for("Santa Claus")`
`=> 635`
`w.total_wait_time()`
`=> 1255`
	waiting.txt:
	`Claus, Santa, 10:32`
	`Smith, John, 5:05`
	`Invalid, 5:15`
	`Claus, Santa, 0:03`

```
class WaitingTime

    def initialize(filename)
        @data = Hash.new 0

        File.foreach(filename) do |line|
            if line =~ /^([A-Z][a-z]+), ([A-Z][a-z]+), (\d+):(\d{2})$/
                name = "#{$2} #{$1}"
                time = $3.to_i * 60 + $4.to_i
                @data[name] += time
            end

        end
    end
    def student_waited_for(student_name)
        @data[student_name]

    end
    def total_wait_time()
        @data.inject 0 do |acc, x|
            acc + x[1]
        end

    end
end
```

5. [10pts] Rust

1 (2pts) You can have two mutable references to the same object.

A. True

B. **False**

2 (2pts) Provide an example of a situation that would result in an error in C but **cannot** occur in Rust. Your answer can either be in the form of a writen explanation or a code snippet.

Dangling pointer (double free, use after free), dereferencing a raw pointer

3 (6pts) The following Rust program has multiple compile errors. Explain the errors at lines 6, 8, and 9.

```
1  fn main() {
2      let mut s1 = String::from("hello");
3      {
4          let s2 = s1;
5          println!("String is {}", s2);
6          s2.push_str(" world!");
7      }
8      s1.push_str(" world!");
9      println!("String is {}", s1);
10 }
```

a) Line 6

`s2` is not mutable.

b) Line 8

`s1` has been moved.

c) Line 9

`s1` has been moved.

6. [12pts] RegExp, FA, CFG

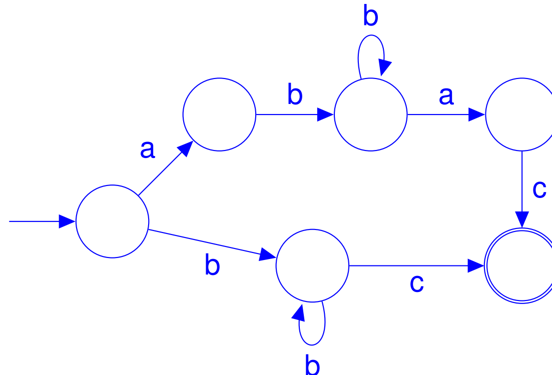
1 (3pts) Write a CFG that matches the language

$$a^x b^y a^x c,\ \text{where } 0 \le x < 2,\ \text{and } y \ge 1$$

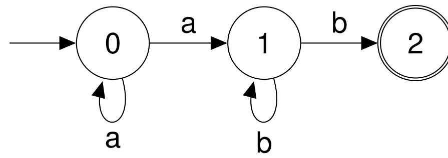
```
S -> Bc | aBac
B -> bB | b
```

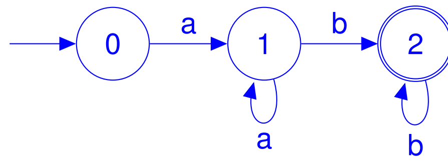
2 (4pts) Can the above CFG be represented by NFA? If yes, draw an NFA representing the above CFG. If not, state why not.

Yes, it can:



3 (5pts) Convert the following NFA to an equivalent DFA:





7. [8pts] Parsing

1 (2pts) Consider the following grammar:

S → **n**S | **n**T**a** | **n**
T → **a**ST | **b**T | **c**

Identify the first sets for each nonterminal in the grammar.

FIRST(S) = **{n}**

FIRST(T) = **{a, b, c}**

2 (6pts) Complete the parser implementation in OCaml below for the grammar on the previous page (shown below for your reference). `lookahead` and `match_tok` are given to you, and tokens are just strings. Do not construct an AST, simply parse the tokens as expected by the grammar and raise the provided exception (`ParseError`) if the input string is invalid.
NOTE: notice that if there are no tokens left, `lookahead` returns an empty string.

S → **n**S | **n**T**a** | **n**
T → **a**ST | **b**T | **c**

```
exception ParseError of string
let tok_list : string list ref = ref []

let lookahead () : string =
    match !tok_list with
    | [] -> ""
    | h::_ -> h
let match_tok (a : string) : unit =
    match !tok_list with
    | h::t when a = h -> tok_list := t
    | _ -> raise (ParseError "bad match")

let rec parse_S () =
    match_tok "n";
    match lookahead () with
    | "n" -> parse_S ()
    | "a" | "b" | "c" -> parse_T ();
                         match_tok "a"
    | "" -> ()
    | _ -> raise (ParseError "parse_S")


and parse_T () =
    match lookahead () with
    | "a" -> match_tok "a";
             parse_S ();
             parse_T ();
    | "b" -> match_tok "b";
             parse_T ();
    | "c" -> match_tok "c";
    | _ -> raise (ParseError "parse_T")
```

8. [10pts] Operational Semantics

1 (4pts) Using the following rules, show that:

`A, x:8 in if 8 > 7 then x - 7 else 7 - x` $\to$ **1**

$$\frac{}{A;\textbf{false}\to\textbf{false}} \qquad \frac{}{A;\textbf{true}\to\textbf{true}}$$

$$\frac{}{A;n\to n} \qquad \frac{A(x)=v}{A;x\to v}$$

$$\frac{A;e_1\to n_1 \quad A;e_2\to n_2 \quad v\ is\ n_1>n_2}{A;e_1>e_2\to v} \qquad \frac{A;e_1\to n_1 \quad A;e_2\to n_2 \quad n_3\ is\ n_1-n_2}{A;e_1-e_2\to n_3}$$

$$\frac{A;e_1\to\textbf{true} \quad A;e_2\to v}{A;\ \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3\to v} \qquad \frac{A;e_1\to\textbf{false} \quad A;e_3\to v}{A;\ \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3\to v}$$

$$\frac{\dfrac{\dfrac{}{A,x:8;8\to 8} \quad \dfrac{}{A,x:8;7\to 7} \quad \textbf{true}\ is\ 8>7}{A,x:8;8>7\to\textbf{true}} \qquad \dfrac{\dfrac{A,x:8(x)=8}{A,x:8;x\to 8} \quad \dfrac{}{A,x:8;7\to 7} \quad 1\ is\ 8-7}{A,x:8;x-7\to 1}}{A,x:8;\ \textbf{if}\ 8>7\ \textbf{then}\ x-7\ \textbf{else}\ 7-x\to 1}$$

2 (2pts) Which ONE of the following is true about operational semantics? Operational Semantics...

A. **Defines how programs execute**

B. Defines how programs compile

C. Defines programs mathematically

D. Defines programs in preconditions and postconditions

3 (4pts) We are writing operational semantics rules for the NAND operator. NAND (short for NOT AND) takes the AND of it's two arguments and then negates it.

A	B	A NAND B
false	false	true
false	true	true
true	false	true
true	true	false

Given the following axioms, write the **operational semantics rules** for the NAND operator. The format for all 4 rules is the same, so just fill in the 4 empty templates below. Furthermore, the following 4 rules are related and may be helpful.

$$\frac{}{A; n \rightarrow n} \qquad \frac{A(x) = v}{A; x \rightarrow v}$$

$$\frac{}{A; \textbf{false} \rightarrow \textbf{false}} \qquad \frac{}{A; \textbf{true} \rightarrow \textbf{true}}$$

$$\frac{A; \boxed{\mathbf{e_1}} \rightarrow \boxed{\textbf{false}} \quad A; \boxed{\mathbf{e_2}} \rightarrow \boxed{\textbf{false}}}{A; \boxed{\mathbf{e_1}}\ \text{NAND}\ \boxed{\mathbf{e_2}} \rightarrow \boxed{\textbf{true}}}$$

$$\frac{A; \boxed{\mathbf{e_1}} \rightarrow \boxed{\textbf{false}} \quad A; \boxed{\mathbf{e_2}} \rightarrow \boxed{\textbf{true}}}{A; \boxed{\mathbf{e_1}}\ \text{NAND}\ \boxed{\mathbf{e_2}} \rightarrow \boxed{\textbf{true}}}$$

$$\frac{A; \boxed{\mathbf{e_1}} \rightarrow \boxed{\textbf{true}} \quad A; \boxed{\mathbf{e_2}} \rightarrow \boxed{\textbf{false}}}{A; \boxed{\mathbf{e_1}}\ \text{NAND}\ \boxed{\mathbf{e_2}} \rightarrow \boxed{\textbf{true}}}$$

$$\frac{A; \boxed{\mathbf{e_1}} \rightarrow \boxed{\textbf{true}} \quad A; \boxed{\mathbf{e_2}} \rightarrow \boxed{\textbf{true}}}{A; \boxed{\mathbf{e_1}}\ \text{NAND}\ \boxed{\mathbf{e_2}} \rightarrow \boxed{\textbf{false}}}$$

9. [8pts] Security

1 (5pts) BEPIS Co. uses the following Ruby code for their company login page:

```
puts "BEPIS Co. Employee Login Terminal"
puts "Please enter your Top-Secret Username:"
username = gets
puts "Please enter your Password:"
password = hash(gets)

results = db.execute "SELECT * FROM Users
                              WHERE Name = '#{username}'
                              AND Password = '#{password}';"

if not results.nil?
    grant_access()
end
```

a) Name the vulnerability that exists in this code

SQL Injection

b) Name a fix that could have prevented this vulnerability

Prepared statements, input validation (e.g. white/blacklisting)

c) Give an example of an input which would exploit this vulnerability

`'; DROP TABLE Users;--` ***(the leading single-quote is the most important part)***

2 (3pts) Name three different types of security vulnerabilites where user input intended to be used as data is treated as code

1. **SQL Injection**
2. **Shell injection**
3. **XSS**
4. **Buffer overflow**