CMSC 330, Fall 2018 — Final

Name _____

Teaching Assistant

Kameron Aaron Danny Chris Michael P. Justin Cameron B. Derek Kyle Hasan Shriraj Cameron M. Alex Michael S. Pei-Jo

Instructions

- Do not start this exam until you are told to do so.
- You have 120 minutes for this exam.
- This is a closed book exam. No notes or other aids are allowed.
- For partial credit, show all your work and clearly indicate your answers.

Honor Pledge

Please copy and sign the honor pledge: "I pledge on my honor that I have not given or received any unauthorized assistance on this examination."

_____

_____

_____

_____

Section	Points
Programming Language Concepts	8
λ-Calculus	8
OCaml	28
Ruby	13
Rust	17
Regexp, FA, CFG	20
Parsing	10
Operational Semantics	6
Security	10
Total	120

1 Programming Language Concepts

1. [1 pts] (T / F) Both Rust and OCaml allow a programmer to manually manage memory.

   Solution. False. ❑

2. [1 pts] (T / F) Reference counting garbage can collect cyclic dead objects.

   Solution. False. ❑

3. [1 pts] (T / F) Traits in Rust are analogous to interfaces in Java.

   Solution. True. ❑

4. [1 pts] (T / F) Languages that perform "bounds checking," like Java, on read or write attempts are less vulnerable to buffer overflow attacks than ones that don't (such as C).

   Solution. True. ❑

5. [2 pts] Circle all of the following programming language features that Ruby has:

   (a) Dynamic Typing (b) Static Typing (c) Lexical Scoping (d) Dynamic Scoping

   Solution. (a), (c) ❑

6. [2 pts] Given the definitions of `fold_left` and `fold_right` below, which function would you use if you were trying to sum a list of one million elements and why? Be brief.

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

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

   Solution. `fold_left` because it's tail recursive. ❑

2 λ-Calculus

1. Circle T if the expressions are α-equivalent and F otherwise.

   (a) [1 pts] (T / F) $(\lambda y.\ y)\ y$ $\lambda y.\ y\ y$

   Solution. False. ❑

   (b) [1 pts] (T / F) $(\lambda y.\ y)\ y$ $(\lambda x.\ x)\ x$

   Solution. False. ❑

2. [2 pts] Write a λ expression that has no normal form.

   Solution. $\Omega = (\lambda x.x\ x)\ (\lambda.x\ x)$ ❑

3. [4 pts] Reduce the λ expression to β-normal form (to be eligible for partial credit, show each reduction step). If already in normal form, write "normal form." If it reduces infinitely, write "reduces infinitely."

$$(\lambda x.\ (\lambda x.x\ x))\ a\ (\lambda y.y)\ c$$

Solution.

$$\begin{aligned}(\lambda x.\ (\lambda x.x\ x))\ a\ (\lambda y.y)\ c &\to_\beta (\lambda x.x\ x)\ (\lambda y.y)\ c\\ &\to_\beta (\lambda y.y)\ (\lambda y.y)\ c\\ &\to_\beta (\lambda y.y)\ c\\ &\to_\beta c\end{aligned}$$

❑

3 OCaml

You may use `map`, `fold_left`, and `fold_right` on any of the following questions.

1. [8 pts] The `unfoldr` function generates a list of values from a function (the generator) and an initial value (the seed). The generator returns `'a * 'b option`, and should be called repeatedly to build up the list. When the generator returns `None`, the list is finished. When it returns `Some`, the first value of the tuple should be added to the list, and the second value of the tuple should be used as the next seed.

   ```
   let f seed = if seed <= 4 then Some (seed, seed + 1) else None
   unfoldr f 1 = [1; 2; 3; 4]
   unfoldr f 3 = [3; 4]
   ```

   Implement `unfoldr` below.

   ```
   let rec unfoldr (f : 'a -> ('b, 'a) option) (seed : 'a) : 'b list =
   ```

   Solution.

   ```
   let rec unfoldr (f : 'a -> ('b, 'a) option) (seed : 'a) : 'b list =
     match f seed with
     | None -> []
     | Some (x, seed') -> x :: (unfoldr f seed')
   ```

   ❑

2. [5 pts] The Iverson bracket function converts a predicate into a function that returns 0 if the predicate is not satisfied and 1 if it is.

   ```
   let even_pred x = (x mod 2) = 0
   let even_iver = iver even_pred
   ```

Given the above, `even_iver 4 = 1` and `even_iver 5 = 0`. Implement `iver` below.

```
let iver (p : ('a -> bool)) : ('a -> int) =
```

Solution.

```
let iver (p : ('a -> bool)) : ('a -> int) =
  (fun x -> if p x then 1 else 0)
```

❑

3. [10 pts] The prime-counting function `prime_count` returns the number of primes less than or equal to n. Assuming there exists a predicate `prime` that determines if a number is prime, implement `prime_count` **without** making `prime_count` itself recursive. You will receieve at most $\frac{1}{2}$ credit if you make `prime_count` itself recursive. (Hint: Use `unfoldr` and `iver` above).

```
let prime_count (n : int) : int =
```

Solution.

```
let prime_count (n : int) : int =
  let zs = unfoldr (fun seed -> if seed <= n then Some (seed, seed + 1) else None) 1 in
  let ps = map (iver prime) zs in
  foldr (+) ps 0
```

❑

4. [5 pts]

```
type 'a non_empty_list =
  | End of 'a
  | Cons of ('a * 'a non_empty_list)
```

Given the above type declaration of `non_empty_list`, write a function `length` that calculates the length of a `non_empty_list l`.

```
length (End 1) = 1
length (Cons ('d', Cons ('u', Cons ('c', End 'k')))) = 4
```

Implement `length` below.

```
let rec length (l : non_empty_list) : int =
```

Solution.

```
let rec length (l : non_empty_list) : int =
  match l with
  | End _ -> 1
  | Cons (_, t) -> 1 + (length t)
```

❑

4 Ruby

The school's rubber duck population has been getting out of hand. You have been tasked with writing software in order to bring it under control. On the next page, implement the class `DuckSorter` that will read a text file and allow you to search for duck's with a given attribute.

1. [7 pts] Implement the `initalize` method that takes a filename then reads and stores the information about the rubber ducks in a data structure of your choice. If a duck's name appears multiple times in the file, the attribute list should be combined together. If a line is malformed, it should be ignored.

2. [2 pts] Implement the `attributes` method that takes a duck's name and returns an array of all that duck's attributes. Order doesn't matter, but every element in this list should be unique. You may assume that the duck exists.

3. [4 pts] Implement the `search` method that takes a string as an argument containing a single attribute and returns an array of rubber duck names where every duck in the array has the given attribute. Order doesn't matter. If no ducks contain the given attribute, an empty array should be returned.

   Here is an example of an input file called `ducks.txt`:

   ```
   name: Fenton, attributes: large, winter
   name: Santa, attributes: winter, red, hat
   name: Edward, attributes: small, glasses
   name: Fenton, attributes: large, blue
   ```

   All names will begin with a capital letter and be followed by lowercase letters. An attribute will be all lowercase and attributes will be separated by a comma and a space. Each line of the text file will have a duck name and a list of attributes. Here is an example of a program using the above methods.

   ```
   a = DuckSorter.new("ducks.txt")
   puts a.get_attributes("Fenton").inspect
   puts a.search("winter").inspect
   puts a.search("large").inspect
   ```

   This will output:

   ```
   ["large", "winter", "blue"]
   ["Fenton", "Santa"]
   ["Fenton"]
   ```

```
class DuckSorter
    def initalize(filename)

        IO.foreach(filename) { |line|









        }


    end

    def get_attributes(name)




    end

    def search(attribute)









    end
end
```

Solution.

```
@ducks = Hash.new([])
IO.foreach(filename) {|line|
    if line =~ /name:([A-Z][a-z]+), attributes:([a-z]+(, [a-z]+)*)/
        @ducks[$1] += $2.split(", ")
        @ducks[$1].uniq!
    end
}
```

❑

Solution.

```
@ducks[name]
```

❑

Solution.

```
arr = []
@ducks.each{ |k, v|
if v.include?(attribute)
arr.push(k)
}
arr
```

❑

5 Rust

1. [3 pts]

```
fn quack(s1 : String) -> () {
    {
        let s2 = &s1;
        let s3 = &s1;
    }
    let s4 = s1;
    let s5 = s4;
    let s6 = &s5;
    let s7 = s5;
    let s8 = &s7;
    // HERE
}
```

Who is the owner of the data from `s1` at `HERE`?

Solution. `s7`. ❑

2. The following programs have an error. Rewrite the code next to the snippet so there is no error.

   (a) [3 pts]

```
fn quack(s1 : String) -> () {
    s1.push_str(" QUACK");
    println!("Duck says: {}", s1);
}
```

Solution.

```
fn quack(mut s1 : String) -> () {
    s1.push_str(" QUACK");
    println!("Duck says: {}", s1);
}
```

❑

   (b) [4 pts]

```
fn shortest(x : &str, y : &str) -> &str {
    if x.len() < y.len() { x } else { y }
}
```

Solution.

```
fn shortest<'a>(x : &'a str, y : &'a str) -> &'a str {
    if x.len() < y.len() { x } else { y }
}
```

❑

3. [7 pts] For the given `Duck` struct, write a function `oldest` that takes an immutable reference to a vector of `Duck` and returns a reference to the oldest one. You may get an `Iterator` from a vector via a call to its `iter` function. You may also assume the input vector is non-empty.

```
struct Duck {
    name: String,
    age: i32
}

fn oldest (ducks : &Vec<Duck>) -> &Duck {









}
```

Solution.

```
fn oldest (ducks: &Vec<Duck>) -> &Duck {
    let mut old = &ducks[0];
    for d in ducks.iter() {
        if d.age > old.age {
            old = d;
        }
    }
    old
}
```

❑

6 Regexp, FA, CFG

1. [3 pts] Write a regular expression that describes all the strings over the alphabet $\{d, u, c, k\}$ where u appears exactly once.

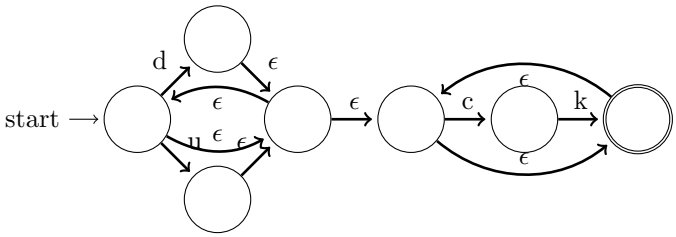
Solution.

$$(d \mid c \mid k)^\star u (d \mid c \mid k)^\star$$

❑

2. [3 pts] Draw an NFA that is equivalent to the regular expression: $(d \mid u)^\star (ck)^\star$

Solution.



❑

3. [5 pts] Write a CFG that accepts strings of the form $d^a uck^b s$ where $a = b + 1$ and $a, b \geq 1$.

Solution.

$$S \to dTs$$

$$T \to dTk \mid uc$$

❑

4. [3 pts] Write a regular expression that accepts the same set of strings as the following CFG. If this is not possible, explain.

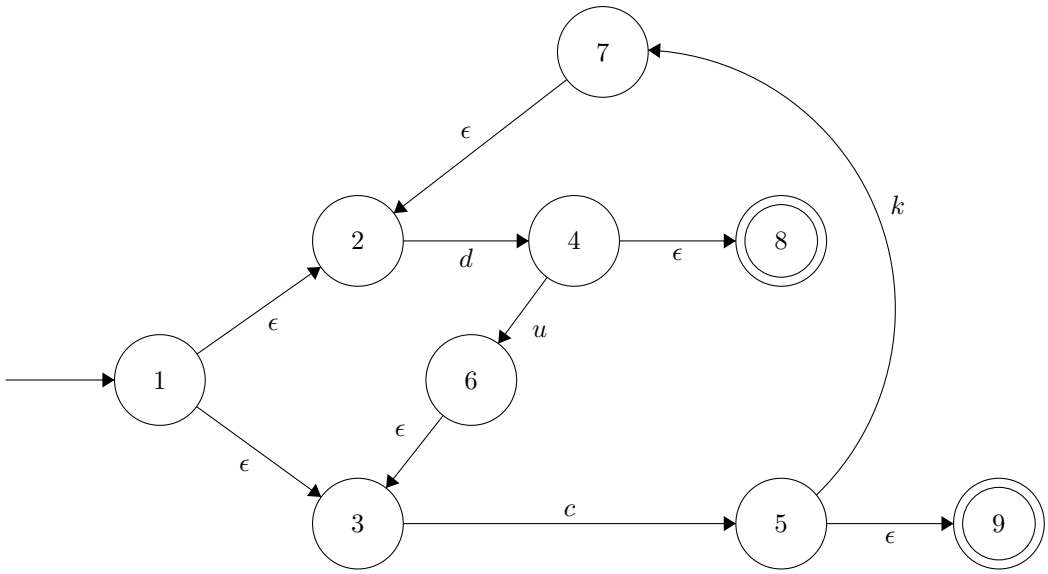
$$S \to AB$$
$$A \to Aquack \mid quack$$
$$B \to X \mid Y$$
$$X \to duX \mid \varepsilon$$
$$Y \to ckY \mid \varepsilon$$

Solution.

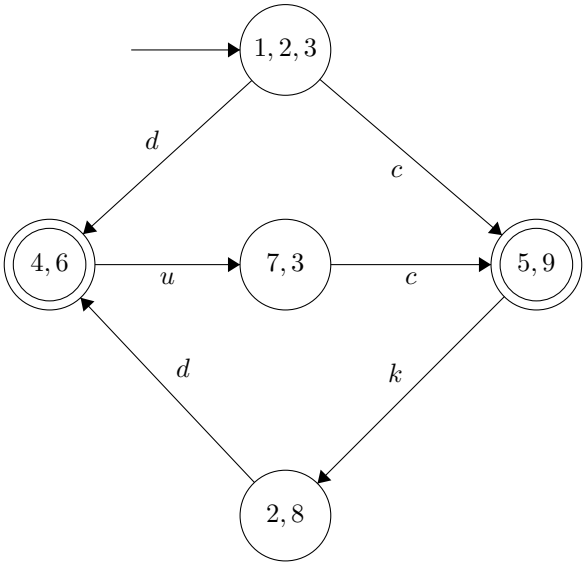
$$(quack)^+((du)^\star \mid (ck)^\star)$$

❑

5. [6 pts] Convert the following NFA to a DFA.



Solution.



❑

7 Parsing

1. Refer to the following ambiguous grammar for the next two questions:

$$S \to S + S \mid a$$

(a) [1 pts] Identify why the grammar is ambiguous (you do not have to prove with derivations).

Solution. There are strings generated by the grammar with multiple left derivation.

$$S \to S + S \to S + S + S \to a + S + S \to a + a + S \to a + a + a$$
$$S \to S + S \to a + S \to a + S + S \to a + a + S \to a + a + a$$

❑

(b) [2 pts] Modify the grammar so that it is no longer ambiguous without changing what strings the language accepts.

Solution. Ambiguity can be resolved by introducing a new nonterminal node.

$$S \to T + S \mid T$$
$$T \to a$$

This happens to be equivalent to a slightly simpler CFG.

$$S \to a + S \mid a$$

❑

2. S-expressions are a notation used to represent list and tree-like data. Refer to the following S-expression grammar for the next two questions:

$$S \to (S.S) \mid [L] \mid n$$
$$L \to SL \mid \varepsilon$$

(a) [2 pts] What are the first sets for the nonterminals in the grammar?

$$\textsc{first}(S) = \{ \qquad \}$$
$$\textsc{first}(L) = \{ \qquad \}$$

Solution.

$$\text{first}(S) = \{(, [, n\}$$
$$\text{first}(L) = \{(, [, n, \varepsilon\}$$

❑

(b) [5 pts] Finish the definition of a recursive descent parser for S-expressions on the next page. You need not build an AST, assume all methods return unit. Note that you are provided the functions `match_tok : string -> unit` and `lookahead : unit -> string`.

For reference, here is the grammar again.

$$S \to (S.S) \mid [L] \mid n$$
$$L \to SL \mid \varepsilon$$

Fill in `parse_S` and `parse_L` below.

```
let lookahead () : string =
  match !tok_list with
  | [] ->  raise (ParseError "no tokens")
  | h::t -> 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 () : unit =
```

```
and parse_L () : unit =
```

Solution.

```
let rec parse_S () =
  match lookahead () with
    | "(" ->
      match_tok "(";
```

```
      parse_S ();
      match_tok ".";
      parse_S ();
      match_tok ")";
    | "[" ->
      match_tok "[";
      parse_L ();
      match_tok "]";
    | "n" ->
      match_tok "n";

and parse_L () =
  match lookahead () with
    | "(" | "[" | "n" ->
      parse_S ();
      parse_L ();
    | _ -> ();
```

❑

8 Operational Semantics

1. [4 pts] Using the rules given below, show: $A;\ \textbf{if}\ 5 > 3\ \textbf{then}\ 3 - 2\ \textbf{else}\ 5 - 1 \Rightarrow 1$

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

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

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

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

$$\frac{}{A;\ \textbf{if}\ 5 > 3\ \textbf{then}\ 3 - 2\ \textbf{else}\ 5 - 1 \Rightarrow 1}$$

Solution.

$$\dfrac{\dfrac{\dfrac{}{A;\ 5 \Rightarrow 5} \quad \dfrac{}{A;\ 3 \Rightarrow 3} \quad 5 > 3\ \text{is}\ \textbf{true}}{A;\ 5 > 3 \Rightarrow \textbf{true}} \quad \dfrac{\dfrac{}{A;\ 3 \Rightarrow 3} \quad \dfrac{}{A;\ 2 \Rightarrow 2} \quad 1\ \text{is}\ 3 - 2}{A;\ 3 - 2 \Rightarrow 1}}{A;\ \textbf{if}\ 5 > 3\ \textbf{then}\ 3 - 2\ \textbf{else}\ 5 - 1 \Rightarrow 1}$$

❑

2. [2 pts]

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

$$\frac{A; e_1 \Rightarrow v_1 \quad A, x : v_1;\ e_2 \Rightarrow v_2}{A;\ \textbf{let}\ x = e_1 \textbf{in}\ e_2 \Rightarrow v_2} \qquad \frac{A;\ e_1 \Rightarrow n_1 \quad A;\ e_2 \Rightarrow n_2 \quad n_3\ is\ n_1 + n_2}{A;\ e_1 + e_2 \Rightarrow n_3}$$

Use the above rules to fill in the given construction:

$$\dfrac{\dfrac{A,\ x=4,\ y=5,\ x=3\ (x)=\boxed{}}{A,\ x=4,\ y=5,\ x=3;\ x\Rightarrow\boxed{}} \qquad \dfrac{A,\ x=4,\ y=5,\ x=3;(y)=5}{A,\ x=4,\ y=5,\ x=3;\ y\Rightarrow 5} \qquad \boxed{}}{A,\ x=4,\ y=5,\ x=3;\ x+y\Rightarrow\boxed{}}$$

Solution.

$$\dfrac{\dfrac{A,\ x=4,\ y=5,\ x=3\ (x)=\boxed{3}}{A,\ x=4,\ y=5,\ x=3;\ x\Rightarrow\boxed{3}} \qquad \dfrac{A,\ x=4,\ y=5,\ x=3;(y)=5}{A,\ x=4,\ y=5,\ x=3;\ y\Rightarrow 5} \qquad \boxed{8\ \textbf{is}\ 3{+}5}}{A,\ x=4,\ y=5,\ x=3;\ x+y\Rightarrow\boxed{8}}$$

❑

9 Security

1. [1 pts] (T / F) A web application is protected from SQL injections by SQL prepared statements.

   Solution. True. ❑

2. [1 pts] (T / F) If a language is statically typed, the compiler prevents buffer overflows.

   Solution. False. ❑

3. [1 pts] (T / F) An XSS vulnerability allows an attacker to execute shell commands on a server.

   Solution. False. ❑

4. [1 pts] Briefly describe how you would prevent an XSS attack.

   Solution. Sanitize user input by replacing special characters with HTML entities. ❑

5. [2 pts] What are the two different types of XSS attack? What is the difference between them?

   Solution.

   - Reflected. A parameter from the URL is displayed on the page allowing for scripts embedded in a URL to run.
   - Stored. Unsanitized user input is stored in the database and displayed back to any requesting user.

   ❑

6. DUCK Corp. has written this fantastic server:

```
puts "Welcome to the DUCK Corp. Terminal."
puts "Do well on your CMSC 330 final to get in."
while true do
   puts "Enter file in the current directory to view: "
```

```
   file = gets.chomp
   if file =~ /\.\./
      print "You can only view files in the current directory."
   else
      `cat #{file}` # runs shell command cat
   end
end
```

(a) [1 pts] Name the vulnerability that exists in this code.

Solution. Shell Injection ❑

(b) [1 pts] Name a fix that could prevent this vulnerability.

Solution. Escaping ❑

(c) [2 pts] Enter a user input that could cause damage to the DUCK Corp.

Solution. `whatever.txt && rm -rf` ❑