CMSC330 Fall 2019 - Midterm 2

First and Last Name (PRINT): ______________________________

9-Digit University ID: ______________________________

Instructions:

- Do not start this test until you are told to do so!
- You have 75 minutes to take this midterm.
- This exam has a total of 100 points, so allocate 45 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.
- **Write your 9-Digit UID at the top of EVERY PAGE.**

1. PL Concepts	/ 15
2. Finite Automata	/ 30
3. CFGs and Parsing	/ 30
4. Operational Semantics	/ 10
5. Lambda Calculus	/ 15
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.**

____

____

____

Signature: ______________________________

1. [15pts] PL Concepts

1 (7pts) **Circle your answers**. Each T/F question is 1 point.

T	F	A regular expression can express all palindromes with letters A-Z, and shorter than 10 letters
T	F	Static analysis, such as type checking, occurs before parsing
T	F	There are multiple paths by which the same string can be accepted in a DFA
T	F	Calling a grammar ambiguous is equivalent to saying a string may have multiple different leftmost derivations
T	F	Using `lookahead` in our parser is an example of predictive parsing
T	F	Operational semantics are analogous to interpreting a program
T	F	Regular expressions are more powerful than DFAs (i.e., they can express more languages than DFAs can)

2 (1pts) The step below is an example of...

(λx . x y) (λz . a z)
(λz . a z) y

A. α-conversion

B. β-reduction

3 (3pts) What is the output of the following OCaml code? (That is, what is printed)

```
let x = ref 0 in
  let y = x in
    y := 1;
    print_int !x;
    print_int !y
```

OUTPUT:

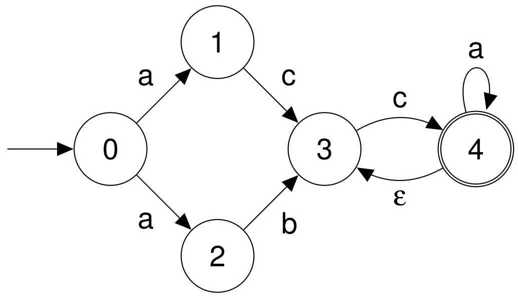
4 (4pts) What is printed by the following OCaml program when the parameters are passed by call-by-name and call-by-value?

```
let f x y =
  if x > 5 then (y,y) else (10,10);;
f 10 (print_string "hello"; 2);;
```

Call-by-name:	**Call-by-value:**

2. [30pts] Finite Automata

1 (6pts) Which of the following strings are accepted by this NFA? *Circle all that apply.*



A. abcab

B. abca

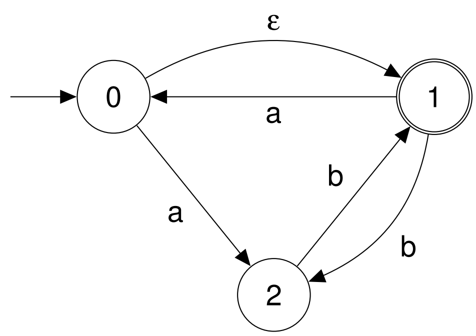
C. abccc

D. aacaccaca

2 (8pts) Construct an NFA that accepts the same language as the following regular expression. There are many answers, any equivalent NFA will be accepted.

`(a+|b*)c?`

3 (6pts) Answer the following questions about this NFA:



e-closure({0}) =	e-closure(move({0, 1}, a)) =

4 (10pts) Give a DFA equivalent to the NFA above. Any equivalent DFA will be accepted, but your answer should be clear. You may give steps for partial credit.

3. [30pts] CFGs and Parsing

1 (5pts) Write a CFG that generates the following language:

$a^x b^y c^{x+y}$, where $x, y \geq 0$

2 (5pts) The following CFG is ambiguous. Rewrite it so that it is not ambiguous. There are many answers, any CFG which is equivalent and is not ambiguous will be accepted. (Note: here, the terminals are: **+**, *****, **(**, **)**, **a**, and **b**.)

$E \rightarrow E\ \mathbf{+}\ E \mid E\ \mathbf{*}\ E \mid \mathbf{(}\ E\ \mathbf{)} \mid \mathbf{a} \mid \mathbf{b}$

3 (4pts) List the FIRST SETS for each nonterminal in the following grammar (lowercase letters are terminals):

$S \rightarrow \mathbf{a}B \mid B\mathbf{b} \mid S\mathbf{c}$
$B \rightarrow \mathbf{d}B \mid \mathbf{d}$

4 (6pts) Indicate if each of the following grammars can be parsed by a recursive descent parser. If the answer is no, give a very brief explanation why.

Grammar	**Yes**	**No**	**If no, why?**
S → S **+** S | N N → **1** | **2** | **3** | **(**S**)**			
S → **a**S | B B → **b**B | **b**			
S → S**b** | A A → **a**A**c** | **c**			

5 (10pts) Complete the OCaml implementation for a recursive-descent parser of the following context-free grammar. The implementation of `match_tok` and `lookahead` are given below:

```
let tok_list = ref [];;
let match_tok x = match !tok_list with
  | h :: t when x = h -> tok_list := t
  | _ -> raise (ParseError "bad match");;
let lookahead () = match !tok_list with
  | [] -> None
  | h :: t -> Some h
```

S → **b**S | **c**T
T → R**a** | R**b**R
R → **d**R | ε

NOTE: this parser takes the imperative approach. Also notice that the tokens are simply strings. So the token list for the string "abcdc" would look like `["a"; "b"; "c"; "d"; "c"]`. You are not creating an AST. If the input is invalid, throw a `ParseError`.

Write your implementation on the next page. The CFG is repeated on the next page for your reference.

```
let rec parse_S () =
    if lookahead () = Some "b" then
        match_tok "b";
        parse_S ()
    else (* fill in below *)
```

S → **b**S | **c**T
T → R**a** | R**b**R
R → **d**R | ε

```
and rec parse_T () = (* fill in below *)
```

```
and rec parse_R () =
    if lookahead () = None then
        ()
    else (* fill in below *)
```

4. [10pts] Operational Semantics

1 (2pts) Below is an incorrect rule for an if-then-else construct when the condition is true. Indentify the mistake, and explain how to fix it. Here, the expression `if a then b else c` is encoded as $\textbf{if-then-else}\ a\ b\ c$.

$$\frac{A; e_1 \to true \qquad A; e_3 \to v}{A; \textbf{if-then-else}\ e_1\ e_2\ e_3 \to v}\ \textsc{IfThenElse-True}$$

2 (3pts) Describe what the operator **myst** does, or give its name.

$$\frac{A; e_1 \to true \qquad A; e_2 \to true}{A; e_1\ \textbf{myst}\ e_2 \to true} \qquad \frac{A; e_1 \to true \qquad A; e_2 \to false}{A; e_1\ \textbf{myst}\ e_2 \to false}$$

$$\frac{A; e_1 \to false \qquad A; e_2 \to true}{A; e_1\ \textbf{myst}\ e_2 \to false} \qquad \frac{A; e_1 \to false \qquad A; e_2 \to false}{A; e_1\ \textbf{myst}\ e_2 \to false}$$

3 (5pts) Using the following rules, show that:

A; `let x = 3 in let x = 2 in x + x` $\rightarrow$ 4

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

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

5. [15pts] Lambda Calculus

1 (8pts) Reduce the expressions as far as possible by showing the intermediate β-reductions and α-conversions. Make sure to show each step for full credit!

(λx. λy. x y) (λy. y) x

(λx. λy. x y y) (λm. m) n

2 (7pts) Reduce the following expression to β-normal form using both call-by-name and call-by-value. Show each step, including any β-reductions and α-conversions. If there is infinite reduction, write "infinite reduction."

$$(\lambda y.x)\ ((\lambda x.\ x\ x\ x)\ (\lambda z.\ z\ z\ z))$$

Call-by-name:

Call-by-value: