CMSC330 Spring 2018 Midterm 2
9:30am/ 11:00am/ 3:30pm

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

__________________________________________________________________

Discussion Time (circle one) 10am 11am 12pm 1pm 2pm 3pm

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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 PL Concepts</td>
<td>/9</td>
</tr>
<tr>
<td>2 Finite Automata</td>
<td>/21</td>
</tr>
<tr>
<td>3 Context Free Grammars</td>
<td>/20</td>
</tr>
<tr>
<td>4 Parsing</td>
<td>/17</td>
</tr>
<tr>
<td>5 Operational Semantics</td>
<td>/10</td>
</tr>
<tr>
<td>6 Lambda Calculus</td>
<td>/13</td>
</tr>
<tr>
<td>7 FP &amp; Objects, Tail Recursion</td>
<td>/10</td>
</tr>
<tr>
<td>Total</td>
<td>/100</td>
</tr>
</tbody>
</table>
1. PL concepts [9 pts]

A. [4 pts] Circle true or false for each of the following (1 point each):

a) True / False  Any language accepted by an NFA can be accepted by a DFA
b) True / False  There are some regexps that do not have a corresponding DFA
c) True / False  Lambda calculus is Turing complete
d) True / False  The Y combinator is used to encode numbers and addition

B. [1 pt] In my SmallC interpreter, the token list for string “1-1” showed up as [Tok_Num 1; Tok_Num -1]. I was expecting [Tok_Num 1, Tok_Minus, Tok_Num 1]. This problem is caused by an error in my (circle the right one):

   a) Interpreter
   b) Lexer
   c) Parser
   d) Type Checker

C. [4 pts] What is printed when evaluating the following expression using the CBV (Call by value) and CBN (Call by name) evaluation strategy?

   (fun x -> x; x) (print_string "hi")

<table>
<thead>
<tr>
<th>CBV</th>
<th>CBN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Finite Automata [21 pts]

A. [4 pts] Consider the following automaton which operates over alphabet \{a,b\}.

Which of the following are true about it (circle the letter of the statement)?

a. (2 pts) It is an DFA
b. (2 pts) It is minimal

B. [5 pts] Which of the following strings are accepted by this automaton? Circle them.

aababaa    abbbaaa    bbba    bbabbaba    aaabbaabba

---

Diagram 1

Diagram 2
C. [6 pts] Draw a finite automaton that accepts the same strings as the regular expression 
(a|b)+| (ab*c)

D. [6 pts] Convert following NFA to a DFA.
3. Context Free Grammars [20 pts]

A. [1 pt] True / False In the following grammar, the + operator is left-associative.

\[
E \rightarrow E + T | E - T | T \\
T \rightarrow a | b | c | (E)
\]

B. [11 pts] Consider the following CFG, in which p and q are terminals, and A and B are nonterminals.

   a. [4 pts] Which of the following strings are accepted? Circle them.

   \[
   A \rightarrow pAq | B \\
   B \rightarrow pB | Bq | pq
   \]

   Circle: \[pppqqq, pqpq, pppq, p\]

   b. [3 pts] Give a regular expression that accepts the same strings as the CFG. If this is not possible, explain why.

   c. [4 pts] Show that the CFG is ambiguous.
C. [4 pts] Change the following CFG to eliminate left recursion

\[
S \rightarrow S \text{ and } S | T \\
T \rightarrow \text{true} | \text{false}
\]

D. [4 pts] Give a CFG that starts with one or more y followed by twice as many x or z. The grammar accepts the following strings (and many others): \text{yxx, yzz, yzx, yxx, yyyzzz, yyyzz, yyyxxxxx, ...}
4. Parsing [17 pts]

A. [2 pts] Circle whether the following are true or false
   a. True / false  Recursive descent parsing works bottom-up
   b. True / false  Recursive descent parsing is a kind of predictive parsing

B. [2 pts] Name two features of a grammar that make it unsuitable for recursive descent parsing.

Now Consider the following context-free grammar (CFG):

\[
S \rightarrow Sc \mid dS \\
A \rightarrow aBA \mid \epsilon \\
B \rightarrow bB \mid c
\]

C. [1 point] Circle the correct answer about the CFG definition for nonterminal B.
   a. B is left recursive
   b. B is right recursive
   c. B is ambiguous
   d. None of the above

D. [4 points] What are the FIRST SETS of each of the nonterminals in the grammar?
E. (8 points) Complete the implementation for a recursive-descent parser for the CFG.

```ocaml
exception ParseError of string

let tok_list = ref [];; (* filled in by scanner *)

let lookahead () =
  match !tok_list with
  | [] -> None
  | (h::t) -> Some h

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

let rec parse_S () =
  if (lookahead() = Some "a") || (lookahead() = Some "c") then
    (parse_A();
     match_tok "c")
  else (* FILL IN - 4 pts *)

and parse_A () = (* FILL IN - 4 pts *)

and parse_B () =
  if lookahead() = Some "b" then
    (match_tok "b";
     parse_B())
  else if lookahead() = Some "c" then
    match_tok "c"
  else raise (ParseError "bad match")
```

```
S → Ac | dS
A → aBA | ε
B → bB | c
```
5. Operational Semantics [10 pts]

A. [3 pts] Describe in English what the operator `myst` does, or give its usual name (you have seen it before).

\[
\begin{align*}
\text{Mystery(1):} & \quad \frac{A; e_1 \Rightarrow \text{true} \quad A; e_2 \Rightarrow \text{true}}{A; e_1 \text{ myst } e_2 \Rightarrow \text{false}} \\
\text{Mystery(2):} & \quad \frac{A; e_1 \Rightarrow \text{false} \quad A; e_2 \Rightarrow \text{false}}{A; e_1 \text{ myst } e_2 \Rightarrow \text{false}} \\
\text{Mystery(3):} & \quad \frac{A; e_1 \Rightarrow \text{true} \quad A; e_2 \Rightarrow \text{false}}{A; e_1 \text{ myst } e_2 \Rightarrow \text{true}} \\
\text{Mystery(4):} & \quad \frac{A; e_1 \Rightarrow \text{false} \quad A; e_2 \Rightarrow \text{true}}{A; e_1 \text{ myst } e_2 \Rightarrow \text{true}}
\end{align*}
\]

B. [3 pts] Below are incorrect rules for conditionals. Circle the key part of each rule that is incorrect. Feel free to explain, for clarity.

\[
\begin{align*}
\text{Bad-If-True:} & \quad \frac{A; e \Rightarrow \text{true}}{A; s_1 \Rightarrow A_1} \\
& \quad \frac{A; s_1 \Rightarrow A_1}{A; s_2 \Rightarrow A_2} \\
& \quad \frac{A; \textbf{if } e \ s_1 \ s_2 \Rightarrow A_1}{A; \textbf{if } e \ s_1 \ s_2 \Rightarrow A_1}
\end{align*}
\]
C. [4 pts] The statement \texttt{s unless e} will execute statement \texttt{s} if \texttt{e} evaluates to \texttt{false} and has no effect if \texttt{e} evaluates to \texttt{true}. Implement the semantics for \texttt{unless} by filling in the boxes below. (Like in SmallC, you can assume that expressions have no effect on the environment.)

```
Unless-True
A; s unless e ⇒ [ ]

Unless-False
A; s unless e ⇒ [ ]
```
6. Lambda Calculus [12 pts]

A. [2 pts] Circle all occurrences of **free variables** in the following λ-term:

\[ \lambda x. z (\lambda y. x y) y x \]

B. [2 pts] Circle whether the following statements are true or false

a. True / False \[ \lambda x. \lambda y. y x \] is alpha-equivalent to \[ \lambda f. \lambda n. f n \]

b. True / False \[ \lambda x. \lambda y. y x \] is alpha-equivalent to \[ (\lambda x. \lambda y. y) x \]

C. Reduce each lambda expression to beta-normal form (to be eligible for partial credit, show each reduction step). If already in normal form, write “normal form.”

a) [2 pts] \( (\lambda z. \lambda y. z) x \)

b) [2 pts] \( (\lambda x. \lambda x. x x) y \)

c) [3 pts] \( (\lambda z. (\lambda x. z) z) (\lambda y. x y) \)

D. [2 pts] Which of the following lambda terms has the same semantics as this bit of OCaml code: (circle exactly one)

```ocaml
let func x = (fun y -> y x) in func a b
```

a) \( (\lambda y. y x) a b \)

b) \( (\lambda x. (\lambda y. y x) a b) \)

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

d) \( (x (\lambda y. y x)) a b \)
7. FP & Objects, Tail Recursion [10 pts]

A. [5 pts] Given the Java class `Point` on the left, write the OCaml encoding of the `Point` class on the right. *(Hint: make() returns a tuple of 3 functions, as shown in the code at the bottom of the righthand-side.)*

```java
class Point {
    private int x=0,y=0;
    void set(int x, int y){
        this.x = x;
        this.y = y;
    }
    int getX(){return x;}
    Int getY(){return y;}
}
Point p = new Point();
p.set(2,6);
int x = p.getX();
```

```ocaml
let make () =

let (set,getx,gety) = make ();;
set 2 6;;
let x = getx ();
```

B. [5 pts] Write a **tail-recursive** version of the `sum` function, which sums all elements of a list, having type `int list -> int`.