CMSC330 - Organization of Programming Languages  
Spring 2023 - Final

CMSC330 Course Staff  
University of Maryland  
Department of Computer Science

Name: ________________________________  
UID: ________________________________  

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination  

Signature: ________________________________  

Ground Rules

• You may use anything on the accompanying reference sheet anywhere on this exam  
• Please write legibly. **If we cannot read your answer you will not receive credit**  
• You may not leave the room or hand in your exam within the last 10 minutes of the exam  
• If anything is unclear, ask a proctor. If you are still confused, write down your assumptions in the margin

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>10</td>
</tr>
<tr>
<td>Q2</td>
<td>7</td>
</tr>
<tr>
<td>Q3</td>
<td>15</td>
</tr>
<tr>
<td>Q4</td>
<td>15</td>
</tr>
<tr>
<td>Q5</td>
<td>12</td>
</tr>
<tr>
<td>Q6</td>
<td>15</td>
</tr>
<tr>
<td>Q7</td>
<td>18</td>
</tr>
<tr>
<td>Q8</td>
<td>8</td>
</tr>
<tr>
<td>EC</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>100 + 5</td>
</tr>
</tbody>
</table>


Problem 1: Language Concepts  [Total 10 pts]

(\lambda x. abx) is alpha-equivalent to (\lambda c. xyc)  
True  False

For statically typed languages, type checking occurs during the parsing phrase  
True  False

Dangling Pointers are prevented in Rust  
True  False

Lifetimes are part of a variable's type in Rust  
True  False

"Missing semicolon on line 12" is an error that would raised during evaluation  
True  False

S → S – S|n is an ambiguous grammar  
True  False

Grammar is a subset of Syntax  
True  False

Mark and Sweep is faster than Reference Counting on average  
True  False

A rust function with the following header will compile: fn myst(a:&str, b:&u32, c:&u32) -> &str  True  False

Ocaml's 'let x = x +1 in x' is operationally the same as Ruby's 'x = x + 1'  True  False

Problem 2: Regex  [Total 7 pts]

(a) Which of the following strings are accepted by the regular expression below?

/\[\lambda \delta \sigma +\omega |\beta \]/

Circle NONE if none of the first five (5) options match.  [3 pts]

\lambda \alpha \beta \delta \delta \omega \lambda \sigma \lambda \beta \omega \beta NONE

(b) Write a regular expression that describes a comma separated integer list of odd length.  [4 pts]

Examples:

Valid  Invalid
1  1,2
1,2,3  1,3
-6,-1,-3


Problem 3: Higher Order Functions

Given the following type, write an expression that matches that type. You may not use type annotations and all pattern matching must be exhaustive. You must use map or fold in your answer.

(a) string list -> string

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

Given the expression, write down it's type. You will need to evaluate it first.

(c) fun a b c -> if a b then [b+1] else c

(d) (fun x -> fun y -> y x) ((fun y -> y + 1) 5)

(e) let c = if true then false else true in fun a -> fun b c -> b c > a c
Problem 4: Finite State Machines

Using the subset algorithm, convert the following NFA to a DFA, and fill in the blanks appropriately matching the DFA provided with the right nodes and transitions. Only the blanks will be graded.

NFA:

DFA:

Final States:

S1 S2 S3 S4 S5
Problem 5: Operational Semantics

Consider the following rules for 2 Languages, using Ruby as the Metalanguage:

<table>
<thead>
<tr>
<th>Language 1</th>
<th>Language 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>true → true</td>
<td>true → true</td>
</tr>
<tr>
<td>false → false</td>
<td>false → false</td>
</tr>
<tr>
<td>$A(x) = v$</td>
<td>$A(x) = v$</td>
</tr>
<tr>
<td>$A; x ⇒ v$</td>
<td>$A; x ⇒ v$</td>
</tr>
</tbody>
</table>

$A; e_1 ⇒ v_1 \quad A; e_2 ⇒ v_2 \quad v_3 = v_1 and v_2$

$A; e_1 & e_2 ⇒ v_2$

$A; let x = e_1 in e_2 ⇒ v_2$

(a) Convert the following Language 1 sentence to it's language 2 counterpart

$A; let x = true in false && x$

(b) Complete the opsem proof for the following program using Language 1:

$let x = true in false && x$

$A; let x = true in false && x ⇒ false$

<table>
<thead>
<tr>
<th>Blank 1:</th>
<th>Blank 2:</th>
<th>Blank 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Blank 4:</th>
<th>Blank 5:</th>
<th>Blank 6:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Blank 7:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Problem 6: Lambda Calculus

Perform a single \( \beta \)-reduction using lazy (call by name) evaluation on the outermost expression. If you cannot reduce it, write Beta Normal Form. Do not \( \alpha \)-convert your final answer.

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

Perform a single \( \beta \)-reduction using Eager (call by value) evaluation on the outermost expression. If you cannot reduce it, write Beta Normal Form. Do not \( \alpha \)-convert your final answer.

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

Convert the following expressions to Beta Normal Form. If it is already in Beta Normal Form, circle BNF. If the answer is not given, circle None.

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

\[\lambda y. y y \quad \lambda y. x x y \quad \lambda a. y y a \quad y y y \quad \text{BNF} \quad \text{infinite recursion} \quad \text{None}\]

(d) \((\lambda x. x x x) (\lambda x. x x x)\)  

\[(\lambda x. x x x) \quad x x x \quad (\lambda x. x x x)(\lambda x. x x x) \quad x \quad \text{BNF} \quad \text{infinite recursion} \quad \text{None}\]

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

\[\lambda x. (\lambda b. a b) \quad (\lambda b. a b) \quad a b \quad \lambda x. a \lambda b. a b \quad \text{BNF} \quad \text{infinite recursion} \quad \text{None}\]
Problem 7: Coding

Consider the following Grammar, where \( n \) is any integer:

\[
S \rightarrow N + S | (N) \\
N \rightarrow n
\]

(a) Ruby Lexer

Write a lexer for this grammar in Ruby, you may use the following as tokens

```ruby
# tokens: n, "Plus", "RParen", "LParen"
# example input-output
lex("2 * -5 + 6") = IOError
lex("2 -7 9 -10") = ["2", ",-7", ",9", ",-10"]
# If an error occurs, you may raise an error
raise IOError.new("Error")
```

def lex(str)
Using the same grammar as before, where \( n \) is any integer:

\[
S \rightarrow N \cdot S | (N) \\
N \rightarrow n
\]

Write a parser for the \( S \) non-terminal in **Ocaml**. You may use the following types and functions:

```ocaml
type tok = Int of int | Plus | RParen | LParen

let lookahead toks = match toks with
  | [] -> None
  | h::t -> Some h

let match_tok toks tok = match toks [] -> raise Error
  | h::t when h = tok -> t
  | _ -> raise Error

(* You may assume raise Error is valid and compiles *)
```

You may assume there is a `parse_n` function of type `tok list -> (tree * tok list)` and that it is correct. The type of `parse_s` is `tok list -> (tree * tok list)`

let rec parse_s toks =
Problem 8: Rust

```rust
fn main() {
    let m = String::from("Hello");
    let t = String::from("World");
    {
        let y = m;
        {
            let c = myfunc(y, t);
            let d = &c;
        }
    }
}
```

fn myfunc<'a>(a: String, b: String) -> String {
    if a.len() > b.len() { a } else { b }
}

Ownership
If there is no owner, write "NONE".

Who is the owner of "Hello" immediately after line 11 is run?

Who is the owner of "World" immediately after line 5 is run?

Lifetimes

What is the last line executed before "Hello" dropped?

What is the last line executed before "World" dropped?

Problem 9: Extra Credit

What is your favorite pun?

Problem 10: Extra Credit

Who is your discussion TA and what is your section number?
You may use this area as scratch space