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>Total</td>
<td>100 + 5</td>
</tr>
</tbody>
</table>
**Problem 1: Language Concepts**

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

\(\text{True} \quad \text{False}\)

For statically typed languages, type checking occurs during the parsing phase

\(\text{True} \quad \text{False}\)

Dangling Pointers are prevented in Rust

\(\text{True} \quad \text{False}\)

Lifetimes are part of a variable's type in Rust

\(\text{True} \quad \text{False}\)

"Missing semicolon on line 12" is an error that would raise during evaluation parser's job. This is called a linter

\(\text{True} \quad \text{False}\)

\(\text{True} \quad \text{False}\)

Mark and Sweep is faster than Reference Counting on average

Stated in class

\(\text{True} \quad \text{False}\)

A rust function with the following header will compile: \(\text{fn myst}(\text{a:&str}, \text{b:&u32}, \text{c:&u32}) \rightarrow &\text{str}\)

\(\text{True} \quad \text{False}\)

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

\(\text{True} \quad \text{False}\)

One makes a new binding to a new variable, the other updates the binding

**Problem 2: Regex**

(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.

<table>
<thead>
<tr>
<th>(\lambda \lambda \beta)</th>
<th>(\delta)</th>
<th>(\delta \omega \lambda)</th>
<th>(\sigma \lambda \beta \beta)</th>
<th>(\omega \beta)</th>
<th>NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The scope of the OR is not restricted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

\(|-?\d+(,-?\d+)*\d+$/\)}
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

\[
\text{fun } a \rightarrow \text{fold (fun } a \rightarrow a \cdot h) \text{ of } a
\]

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

\[
\text{fun } a \ b \ c \ d \rightarrow \text{map } d \text{ (fold } c \text{ of } a \ b)
\]

Given the expression, write down its type. **You will need to evaluate it first**

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

\[
\text{(int -> bool) -> int -> int list -> int list}
\]

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

\[
\text{(int -> a) -> a}
\]

(e) let c = if true then false else true in fun a -> fun b c -> b c > a c

\[
\text{('a -> 'b) -> ('a -> 'b) -> 'a -> bool}
\]

The first 'let c = ...' is useless since the second fun will rebind c to an input
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:

E1: \( c \)
E2: \( b|a \)
E3: \( a|b \)
E4: \( b \)
E5: \( b \)
E6: \( a \)
E7: \( b \)
E8: \( a \)

E2 and E3 could be swapped

Final States:

\[
\begin{align*}
S1 & : 1 \\
S2 & : 2 \\
S3 & : 4,5,6 \\
S4 & : 3 \\
S5 & : 3,4,5,6 \\
\end{align*}
\]
Problem 5: Operational Semantics

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

Language 1:

true → true
false → false
A(x) = v
A; x ⇒ v

A; e1 ⇒ v1
A; e2 ⇒ v2
A; e1 && e2 ⇒ v3

A; e1 ⇒ v1
A; x : v1; e2 ⇒ v2
A; let x = e1 in e2 ⇒ v2

Language 2

true → true
false → false
A(x) = v
A; x ⇒ v

A; e1 ⇒ v1
A; e2 ⇒ v2
v3 = v1 && v2

A; (λx.λy. x y x) e1 e2 ⇒ v3
A; (λ x : e1) e2 ⇒ v2

(a) Convert the following Language 1 sentence to its Language 2 counterpart

A; let x = true in false && x

A; (λ x: (λx.λy. x y x) false) true

(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

Blank 1: A; true ⇒ true
Blank 2: A; x: true; false ⇒ false
Blank 3: A; x: true(x) = true
Blank 4: A; x: true; x ⇒ true
Blank 5: false = false and false
Blank 6: A; x: true; false && x
Blank 7: false
**Problem 6: Lambda Calculus**

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

(a) \((a \lambda x. x a)(\lambda y. y y)\) \[3 pts\]

**Beta Normal Form**

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

(b) \((\lambda x. a b c)((\lambda x. (x x)) x)\) \[3 pts\]

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

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) \((\lambda x. \lambda y. x y)((\lambda b. b b) y)\) \[3 pts\]

\[
\begin{array}{cccccccc}
\lambda y. y y & y y & BNF & \infty & None
\end{array}
\]

(d) \((\lambda x. x x)(\lambda x. x x x)\) \[3 pts\]

\[
\begin{array}{cccccccc}
\lambda x. x x & x x & (\lambda x. x x x)(\lambda x. x x x) & x & BNF & \infty & None
\end{array}
\]

(e) \(\lambda x. (\lambda a. a b)(\lambda b. a b)\) \[3 pts\]

\[
\begin{array}{cccccccc}
\lambda x. (\lambda a. a b)(\lambda b. a) & a b & (\lambda x. a \lambda b. a b) & BNF & \infty & None
\end{array}
\]
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)
```
(b) Ocaml Parser

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
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)`

```ocaml
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?
- a

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

Lifetimes

What is the last line executed before "Hello" dropped?
- 12/13

What is the last line executed before "World" dropped?
- 6/7

Problem 9: Extra Credit

What is your favorite pun?
I'm not a programmer, I'm pro-grammar

Problem 10: Extra Credit

Who is your discussion TA and what is your section number?
Better question: who was your favorite TA?
You may use this area as scratch space