CMSC330 - Organization of Programming Languages
Fall 2024 - 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

- Please write legibly. **If we cannot read your answer you will not receive credit.**
- You may use anything on the accompanying reference sheet anywhere on this exam
- Please remove the reference sheet from the exam
- The back of the reference sheet has some scratch space on it. If you use it, you must turn in your scratch work
- 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

Question	Points
P1	10
P2.	10
P3.	15
P4.	6
P5.	4
P6.	6
P7.	6
P8.	6
P9.	10
P10.	3
P11.	8
P12.	16
Total	100

Problem 1: Concepts

[Total 10 pts]

	True	False
Some Buffer Overflow vulnerabilities can be prevented by having a type-safe type system	T	F
If you are at some state B in an FSM, the history of your path determines where you go next	T	F
Context Free Grammars can recognize strings with balanced parenthesis	T	F
The best case runtime of the tokenize function from project 4 is polynomial.	T	F
Stop and Copy Garbage collection will not clean up cyclic data structures	T	F
The rules of references that Rust uses help prevents double frees	T	F
Both Ocaml and Rust are statically typed	T	F
In Rust, null pointer exceptions will not occur	T	F
OCaml is Turing complete which means it can solve more problems than Rust	T	F
If a language is well-typed, it must also be well-defined	T	F

Problem 2: Regex

[Total 10 pts]

If you run `ping google.com -c 2` on the command line, you get an echo response from your destination sending 2 network packets. It can be useful to see if you have an active internet connection or if a website is down. An example `ping` command can return:

```
PING google.com (192.168.255.255) 56 bytes of data
64 bytes from 192.168.255.255: icmp_seq=1 ttl=60 time=8.57 ms
64 bytes from 192.168.255.255: icmp_seq=2 ttl=60 time=2.61 ms
--- google.com ping statistics ---
2 packet transmitted, 2 received, 0% packet loss, time 1003ms
```

Write a regex that describes each part of this echoed response:

(a) IP Address [2 pts]

IP Addresses appear multiple times in the following lines. 192.168.255.255 is an IP address. Valid IP addresses will look like 'xxx.xxx.xxx.xxx' where 'xxx' is from 0-255 (inclusive). **Write the regex for the 'xxx' part (the range of 0-255, inclusive).** This regex will be used in the later parts as **IP**.

IP =

(b) Destination Summary - **This is the first line in the example** [2 pts]

Example: `PING google.com (192.168.255.255) 56 bytes of data`

It summarizes where you are pinging and how many bytes the sent data will be. The domain name will be **at least one lowercase letter** followed by any number of dots (.) and lowercase letters with no consecutive dots (it may end in a dot). Only 56 or 64 bytes of data will be sent.

PING ____________________ \\(**IP**(\\.**IP**){3}\\) __________ bytes of data

(c) Received Data - **The second and third lines in the example are instances of this** [2 pts]

Example: `64 bytes from 192.168.255.255: icmp_seq=1 ttl=60 time=8.57 ms`
Example: `64 bytes from 192.168.255.255: icmp_seq=2 ttl=60 time=2.61 ms`

Received data will receive 32 or 64 bytes from an IP address. icmp_seq is the sequence number of the packet (≥ 0), ttl will also be ≥ 0. Time will be any number ≥ 0 with 2 digits after the decimal. You will not need to check if the sequence is in order, or if the IP address is consistent across responses.

________ bytes from \\(**IP**(\\.**IP**){3}\\): icmp_seq= ________ ttl= ________ time=__________ms

(d) Statistics - **The last line in the example** [3 pts]

Example: `2 packet transmitted, 2 received, 0% packet loss, time 1003ms`

The statistics line will have how many packets were transmitted (≥ 0), how many were received (≥ 0), the percent packet loss ($0 - 100$ inclusive), and the time (≥ 0). Time in this response does not include decimals. You don't need to check if the math is right.

________ packets transmitted, ________ received, ____________% packet loss, time ________ms

Problem 3: FSM

[Total 15 pts]

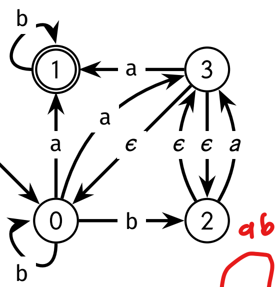
(a) Convert the below NFA to a DFA.
Draw a **box** around your final answer.

[10 pts]

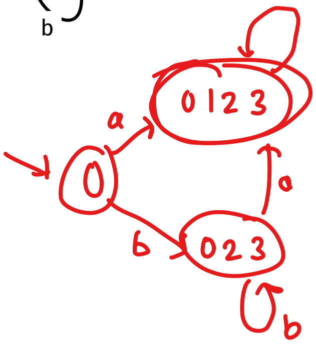
1, 3, 0, 2

0, 1, 2, 3

Scratch Space:



R	a	b
0	0123	0, 2, 3
0123	0123	0123
0, 2, 3	0, 1, 2, 3	0, 2, 3



(b) Write a **CFG** that describes strings accepted by the NFA above.

[5 pts]

Problem 4: OCaml Typing

[Total 6 pts]

Give the type of the function `'foo'`. If there is a type error, put "ERROR"

```
let foo a = fun b -> map a (map b [1;2;3])
```

```
let foo a b c -> if a (b c) then
                     c
                 else
                     c
```

Problem 5: Evaluation

[Total 4 pts]

Evaluate the following OCaml expressions. It there is a compilation error, put "ERROR"

```
let foo a = fun b -> map a (map b [1;2;3]) in
foo (fun x -> -x) (fun y -> y * 4)
```

```
let foo a b c -> if a (b c) then
                     c
                 else
                     c
  in foo (fun a -> 0 ) (fun b -> b) true
```

Problem 6: Property Based Testing

[Total 6 pts]

Consider an attempted (buggy!) implementation of the `double_up_and_dance` function from project 6. `double_up_and_dance` *should* create a vector that contains duplicates of each item in the input slice, with the second item in the input slice added to the fifth element of the created vector, if the created vector is longer than 4 elements. It then returns the created vector. Examples:

```
double_up_and_dance(&[1, 2, 3]) // returns vec![1, 1, 2, 2, 5, 3]
double_up_and_dance(&[1, 2]) // returns vec![1, 1, 2, 2]

pub fn double_up_and_dance(slice: &[i32]) -> Vec<i32> {
    let mut ret = Vec::with_capacity(slice.len() * 2);
    for &elem in slice {
        ret.push(elem);
        ret.push(elem);
    }
    if let Some(val) = ret.get_mut(5) {
        *val += slice[0];
    }
    ret
}
```

Consider the property *p*:

The 5th element of the `double_up_and_dance` vector will be greater than the 2nd element of the input slice (if they exist).

Is p a valid property? Yes No

Suppose we wanted to write this test. We would encode the property as the following:

```
fn test_prop(v in prop::collection::vec(usize,1..10)){
    //will generate random usize vectors of lengths 1 through 10
    if v.len() < 3{
        assert!(true)
    }else{
        assert!(double_up_and_dance(&v).get(5).unwrap() > v.get(2).unwrap())
    }
}
```

Is `test_prop` a correct encoding of the property p? Yes No

If we test this property on the provided implementation of `double_up_and_dance`, will it ever assert false?

Y Yes

N No

Problem 7: Interpreters

[Total 6 pts]

Given the following CFG, at what stage of language processing would each expression **fail**?
Mark **'Valid'** if the expression would be accepted by the grammar and evaluate properly. Assume the only symbols allowed are those found in the grammar. Choose only one choice for each expression.
Note: For expressions that result in an infinite loop, consider them to fail at the evaluator step.

```
E -> LX.E | E E | (E) | X
X -> c
```

For all $c \in X$, c is a lowercase English character

	Lexer	Parser	Evaluator	Valid
`LZ.a`	L	P	E	V
`Lx.(Ly.x y)`	L	P	E	V
`x ((Ld.e y) e`	L	P	E	V
`(La.(Ly.(e e) Ly.(y y)))`	L	P	E	V
`(Lt.(Lx.(t y x)).z)`	L	P	E	V
`(Lx.x x) (Lx.x x)`	L	P	E	V

Problem 8: Type Checking

[Total 6 pts]

Consider the following Typing Rules for Ocaml:

$$\frac{}{G \vdash \text{true} : \text{bool}} \qquad \frac{}{G \vdash \text{false} : \text{bool}} \qquad \frac{}{G \vdash n : \text{int}}$$

$$\frac{}{G \vdash x : G(x)} \qquad \frac{G \vdash e_1 : int \qquad G \vdash e_2 : int \qquad + = (int, int, int)}{G \vdash e_1 + e_2 : int}$$

$$\frac{G \vdash e_1 : t_1 \qquad G, x : t_1 \vdash e_2 : t_2}{G \vdash let\ x = e_1 \text{ in } e_2 : t_2} \qquad \frac{G \vdash e_1 : bool \qquad G \vdash e_2 : t \qquad G \vdash e_3 : t}{G; if\ e_1\ then\ e_2\ else\ e_3 : t}$$

Write a type-checking proof for the expression

let x = true in if x then 4 else 5 + 7

Problem 9: Lambda Calculus

[Total 10 pts]

(a) Reduce [6 pts]

Reduce the following lambda expression. Show every step. (If applicable, use Eager Evaluation).

$$(\lambda c.\ c\ d\ (\lambda a.\ a))(\lambda b.\ (\lambda c.\ (\lambda a.\ d\ b)))$$

(b) Free Variables: [2 pts]

Circle the free variables in the expression below:

$$(\lambda a.(\lambda c.ab))((\lambda b.(\lambda a.b))(aa(\lambda a.c)))$$

(c) Alpha Equivalence: [2 pts]

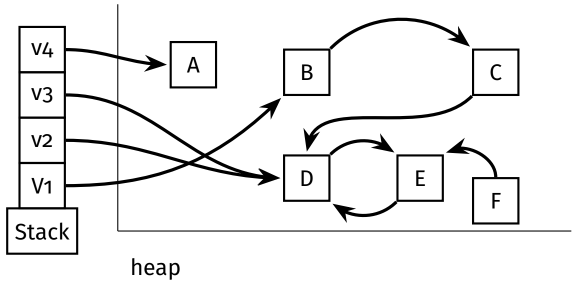
Which of the following are alpha equivalent to the expression $(\lambda b.(\lambda a.b)(ca))$? Select all that apply.

- (A) $(\lambda a.(\lambda a.a)(ca))$
- (B) $(\lambda b.(\lambda c.b)(ca))$
- (C) $(\lambda z.(\lambda y.z)(xy))$
- (D) $(\lambda a.(\lambda b.a)(ca))$

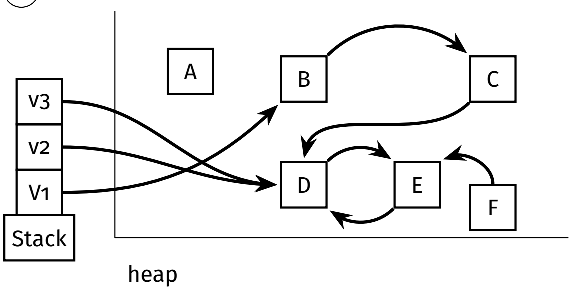
Problem 10: Garbage Collection

[Total 3 pts]

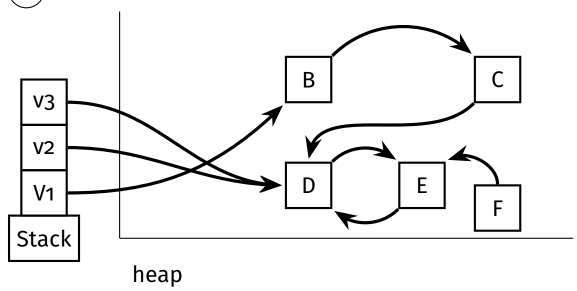
Suppose you have the following memory diagram. Select the correct representation of the stack and heap after the mark and sweep garbage collection technique is run after popping v4 off the stack.



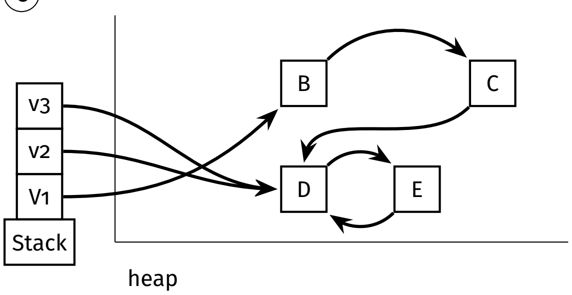
Ⓐ



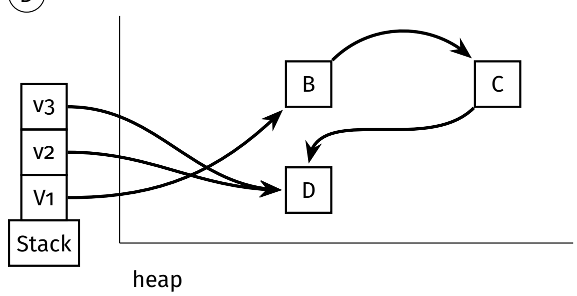
Ⓑ



Ⓒ



Ⓓ



Ⓔ None of the above

Problem 11: Ownership and Lifetimes

[Total 8 pts]

```
fn main(){
    let mut x = 4;
    let y = &mut x;
    println!("{x},{y}");
}
```

Does the code compile? (Y) Yes (N) No

If **no**, explain why not in one sentence:

```
fn main(){
    let x = String::from("bye");
    let y = String::from("farewell");

    let mut z = &x;
    z = &y;
    println!("{x},{z}")
}
```

Does the code compile? (Y) Yes (N) No

If **no**, explain why not in one sentence:

```
fn main(){
    let x = String::from("hello");
    let mut y = x;
    y.push_str(" world!");
    println!("{y}");
}
```

Does the code compile? (Y) Yes (N) No

If **no**, explain why not in one sentence:

```
struct Thing{
    v1:usize,
    v2:String
}

fn main(){
    let mut x = Thing{v1:4,v2:String::from("hi")};
    let y = &mut x;
    y.v2.push_str(" bye");
    let z = y.v2;
    println!("{},{z}",x.v1)
}
```

Does the code compile? (Y) Yes (N) No

If **no**, explain why not in one sentence:

Problem 12: Coding

[Total 16 pts]

(a) OCaml [8 pts]

Given an `int list list`, return true if the first column (the first element in each list) is in descending order, and false otherwise. Assume the matrix is guaranteed to have equal height and width (m x m matrix) and non-empty (m > 0). If items are equal, that is not descending (i.e. a column of [1;1;0] is not descending).

Making this function recursive is optional. If you are not using recursion, you can ignore the *rec* keyword. You can use standard library functions if you wish.

```
ex.                       ex
[[1;2;3;];                   [[7;8;9];
 [4;5;6]; => false            [4;5;6]; => true
 [7;8;9]]                     [1;6;3]]

let rec descending mtx =
```

(b) Rust [8 pts]

Given an `Vec<Vec<u32>>`, return true if the first column (the first element in each list) is in descending order, and false otherwise. Assume the matrix is guaranteed to have equal height and width (m x m matrix) and non-empty (m > 0). If items are equal, that is not descending (i.e. a column of [1;1;0] is not descending).

```
ex.                              ex
vec![vec![1,2,3],                   vec![vec![7,8,9],
     vec![4,5,6], => false               vec![4,5,6], => true
     vec![7,8,9]]                        vec![1,6,3]]

fn descending(v:Vec<Vec<u32>>)->bool{
```

Cheat Sheet

OCaml

```
(* Map and Fold *)
(* ('a -> 'b) -> 'a list -> 'b list *)
let rec map f l = match l with
    [] -> []
   |x::xs -> (f x)::(map f xs)

(* ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a *)
let rec fold_left f a l = match l with
    [] -> a
   |x::xs -> fold_left f (f a x) xs

(* ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b *)
let rec fold_right f l a = match l with
    [] -> a
   |x::xs -> f x (fold_right f xs a)



(* OCaml Function Types *)
::  -: 'a -> 'a list -> 'a list

@ -: 'a list -> 'a list -> 'a list

+, -, *, /   -: int -> int -> int
+., -., *., /. -: float -> float -> float

&&, ||  -: bool -> bool -> bool
not -: bool -> bool

^  -: string -> string -> string

=>,>,=,<,<= :-  'a -> 'a -> bool
```

```
(* Regex in OCaml *)
Re.Posix.re: string -> regex
Re.compile: regex -> compiled_regex

Re.exec: compiled_regex -> string -> group
Re.execp: compiled_regex -> string -> bool
Re.exec_opt: compiled_regex -> string -> group option

Re.matches: compiled_regex -> string -> string list

Re.Group.get: group -> int -> string
Re.Group.get_opt: group -> int -> string option
```

Structure of Regex

$$
\begin{array}{rcl}
R & \rightarrow & \emptyset \\
 & | & \sigma \\
 & | & \epsilon \\
 & | & RR \\
 & | & R|R \\
 & | & R^* \\
\end{array}
$$

Regex

*	zero or more repetitions of the preceding character or group
+	one or more repetitions of the preceding character or group
?	zero or one repetitions of the preceding character or group
.	any character
$r_1 \| r_2$	r_1 or r_2 (eg. a|b means 'a' or 'b')
[abc]	match any character in abc
[ˆr_1]	anything except r_1 (eg. [ˆabc] is anything but an 'a', 'b', or 'c')
[r_1-r_2]	range specification (eg. [a-z] means any letter in the ASCII range of a-z)
{n}	exactly n repetitions of the preceding character or group
{n,}	at least n repetitions of the preceding character or group
{m,n}	at least m and at most n repetitions of the preceding character or group
ˆ	start of string
$	end of string
(r_1)	capture the pattern r_1 and store it somewhere (match group in Python)
\d	any digit, same as [0-9]
\s	any space character like \n, \t, \r, \f, or space

NFA to DFA Algorithm (Subset Construction Algorithm)

NFA (input): $(\Sigma, Q, q_0, F_n, \delta)$, DFA (output): $(\Sigma, R, r_0, F_d, \delta_n)$

$R \leftarrow \{\}$
$r_0 \leftarrow \epsilon - \text{closure}(\sigma, q_0)$
while $\exists$ an unmarked state $r \in R$ **do**
    mark r
    for all $a \in \Sigma$ **do**
        $E \leftarrow \text{move}(\sigma, r, a)$
        $e \leftarrow \epsilon - \text{closure}(\sigma, E)$
        if $e \notin R$ **then**
            $R \leftarrow R \cup \{e\}$
        end if
        $\sigma_n \leftarrow \sigma_n \cup \{r, a, e\}$
    end for
end while
$F_d \leftarrow \{r \mid \exists s \in r \text{ with } s \in F_n\}$

Rust

```
// Vectors
let vec = Vec::new();  // makes a new vector
let vec1 = vec![1,2,3]

vec.push(ele);   // Pushes the element 'ele'
                 // to end of the vector 'vec'

vec.get(x);    // returns the xth index of the
               // vec in an option, with Some(ele)
               // Some(ele) if it exists
               // and None if it doesn't exist
// Strings
let string = String::from("Hello");

string.push_str(&str); // appends the str
                       // to string

vec.to_iter();   // returns an iterator for vec

string.chars()   // returns an iterator of chars
                 // over the a string

iter.rev();         // reverses an iterators direction

iter.next();        // returns an Option of the next
                    // item in the iterator.

struct Building{    // example of struct
    name:String,
    floors:i32,
    locationx:f32,
    locationy:f32,
}

enum Option<T>{ Some(T); None } //enum Option type
option.unwrap();    // returns the item in an Option or
                    // panics if None

if let x = option{...} // assigns x to the element
                       // in the option if it is Some
```