CMSC330 Fall 2023 Quiz 3

Proctoring TA: ____________________ **Name:** ____________________

Section Number: ____________________ **UID:** ____________________

Problem 1: Context Free Grammars

[Total 8 pts]

Consider the following Grammar:

```
E -> aSSc
S -> aSb|bSc|T
T -> a|b|c
```

(a) Is this an ambiguous grammar? [2 pts]

● A Yes ○ B No

(b) If you believe it to be ambiguous, prove it, otherwise derive "aaabbc" [6 pts]

E -> aSSc -> aaSbSc -> aaTbSc -> aaabSc -> aaabTc -> aaabbc
E -> aSSc -> aTSc -> aaSc -> aaaSbc -> aaaTbc -> aaabbc

Problem 2: Lexing Parsing and evaluating

[Total 6 pts]

Given the following CFG, and assuming strong, static typing as is used in **OCaml**, at what stage of language processing would the nearby expressions fail? Mark 'Valid' if the expression would be accepted by the grammar and type checker.

$$E \to M \text{ and } E \mid M \text{ or } E \mid M$$
$$M \to N + M \mid N - M \mid N$$
$$N \to 1 \mid 2 \mid 3 \mid 4 \mid \texttt{true} \mid \texttt{false} \mid (E)$$

Hint: Pay careful attention to the terminal symbols allowed in the grammar.

	Lexer	Parser	Evaluator	Valid
`1 + 2 - (true and false)`	○ L	○ P	● E	○ V
`true + (3 - 2}`	● L	○ P	○ E	○ V
`3 * 1 - 2`	● L	○ P	○ E	○ V
`2 - 1 + 4`	○ L	○ P	○ E	● V
`)(2 or + -`	○ L	● P	○ E	○ V
`true`	○ L	○ P	○ E	● V

Problem 3: OCaml Higher Order Functions [Total 6 pts]

Complete the skeleton code below which defines a simplified version of `partition` which takes a single ”pivot value” and a list. It returns a pair of lists, the first with elements below the pivot value, the second with elements equal to or above the pivot value. The lists returned can have elements from the original list in any order (forward, reverse, other).

```
(* Definition for fold_left *)
let rec fold_left f a lst =
  match lst with
  [] -> a
  |x::t -> fold_left f (f a x) t
```

EXAMPLES:

```
# let partition pivot lst = ...;;
val partition : ’a -> ’a list -> ’a list * ’a list = <fun>

# partition 5 [12; 2; 9; 7; 6; 5; 1; 4];;
- : int list * int list =
([4; 1; 2],    [5; 6; 7; 9; 12])
 (* below 5 ... equal/above 5 *)

# partition "c" ["banana"; "grape"; "carrot"; "pear"; "apple"];;
- : string list * string list =
(["apple"; "banana"],   ["pear"; "carrot"; "grape"])
(* below "c"     ...      equal/above "c" *)
```

```
(*’a -> ’a list -> (’a list * ’a list) *)
let partition pivot lst =
  let helper acc x = match acc with (a,b) -> if x < pivot then (x::a, b) else (a, x::b)

  in fold_left helper ([], []) lst
```