

CMSC330 Fall 2023 Quiz 3



Proctoring TA: _____ Name: _____

Section Number: _____ UID: _____

Problem 1: Context Free Grammars

[Total 8 pts]

Consider the following Grammar:

$E \rightarrow aSSc$
 $S \rightarrow aSb \mid bSc \mid T$
 $T \rightarrow a \mid b \mid c$

(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]

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.

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

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

	Lexer	Parser	Evaluator	Valid
$1 + 2 - (\text{true and false})$	<input type="radio"/> (L)	<input type="radio"/> (P)	<input type="radio"/> (E)	<input type="radio"/> (V)
$\text{true} + (3 - 2)$	<input type="radio"/> (L)	<input type="radio"/> (P)	<input type="radio"/> (E)	<input type="radio"/> (V)
$3 * 1 - 2$	<input type="radio"/> (L)	<input type="radio"/> (P)	<input type="radio"/> (E)	<input type="radio"/> (V)
$2 - 1 + 4$	<input type="radio"/> (L)	<input type="radio"/> (P)	<input type="radio"/> (E)	<input type="radio"/> (V)
$) (2 \text{ or } + -$	<input type="radio"/> (L)	<input type="radio"/> (P)	<input type="radio"/> (E)	<input type="radio"/> (V)
true	<input type="radio"/> (L)	<input type="radio"/> (P)	<input type="radio"/> (E)	<input type="radio"/> (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).

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 *)

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

# 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 =
```

```
in fold_left helper _____ lst
```