On my honor, I have not given, nor received, nor witnessed any unauthorized assistance on this work.

Print name and sign: _____

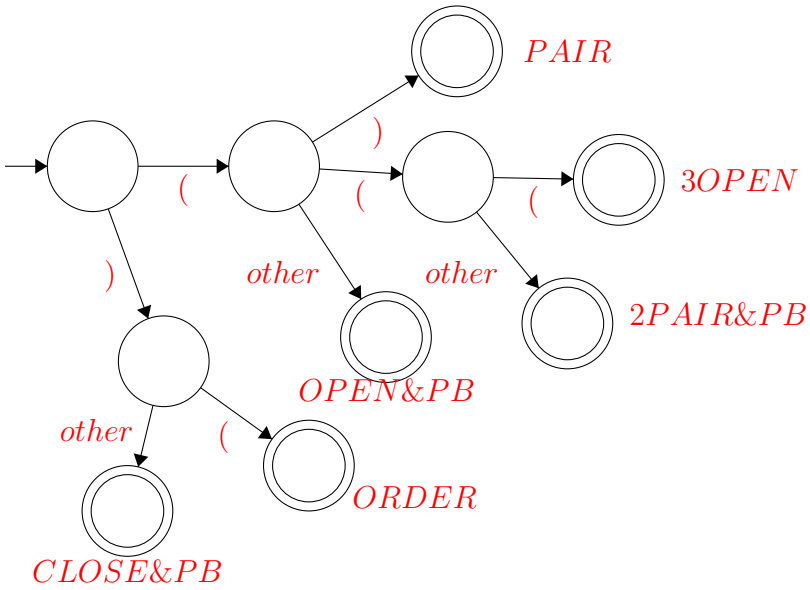
Question:	1	2	3	Total
Points:	10	12	8	30
Score:				

1. (10 points) Construct a transition diagram for a lexical analyzer that recognizes the following patterns (left hand column) and associated tokens (right hand column):

Pattern	**Token Name**
`(`	`OPEN`
`)`	`CLOSE`
`)(`	`ORDER`
`()`	`PAIR`
`((`	`2OPEN`
`(((`	`3OPEN`

Label each arc with the symbol that triggers its transition and each accepting state with the token name (from the right hand column) it recognizes. Note the places where a character must be pushed back on the input stream.

Solution:



Common mistakes:
not indicating which states/transitions require a pushback.
having a final state with a transition OUT of it into another final state.

2. Consider the following grammar:

```
E -> T E'
E' -> + T E' | ε
T -> F T'
T' -> * F T' | ε
F -> (E) | digit
digit -> 0 | 1 | 2 | ... | 9
```

(a) (9 points) Construct a parse tree for the sentence `9 + (0 * 1)`.

Solution:

(b) (3 points) Give 3 other, different strings which are defined by this grammar and have a different form than part a (in other words, not of the form $x + (y * z)$. You do not need to draw parse trees for them.

Solution: Answers vary but could be as simple as a digit (derived from `E->T->F->digit`. Other examples are `(x)` where x is any digit. Another example is `x + y`. Giving three strings with the same form will not receive full credit (eg. listing the strings "8+4, 6+9, and 1+3" would receive only 1 point of credit).

3. Consider the following context-free grammar:

```
S -> S and S | S or S | (S) | true | false
```

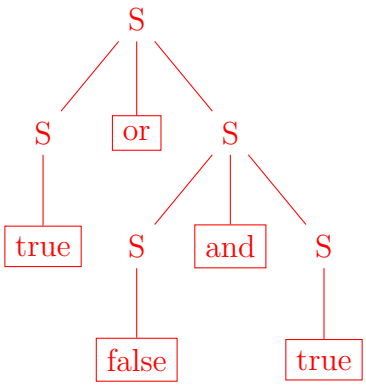
(a) (2 points) List the **terminal(s)**: and, or, (,), true, false.

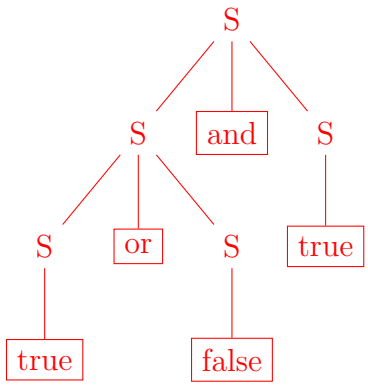
(b) (2 points) List the **non-terminal(s)**: S.

(c) (4 points) A grammar can be ambiguous or unambiguous. The above grammar is ambiguous. Provide an example (and associated parse trees) which prove this fact.

Solution:

Ambiguous grammars mean that 1 statement can be produced multiple ways (different applications of the production rules). Consider the statement `true or false and true` and the two following parse trees:





Common errors: listing production rules for non-terminals.