1. Consider the following grammar:

```
start ::= stmt
stmt ::= ref = expr
ref ::= id
ref ::= ref . id
expr ::= term
expr ::= term + expr
term ::= const
term ::= ref
```

What is the follow set for the nonterminal ref?

2. Explain why the grammar is not LL(1). Rewrite the grammar to be LL(1).
3. Show the DFA that would be constructed by an LR parser for the following grammar. Number your states

start ::= stmt
stmt ::= ref = expr
ref ::= id
expr ::= term
expr ::= term + expr
term ::= const
term ::= ref
4. Using your DFA from question 3, show the sequence of shift and reduce moves that would occur in recognizing the input $x = y + 3$
5. Draw a picture of the various symbol tables that would exist while parsing the body of the procedure `bar` in the following; show in which symbol table each variable name would be found. Then, describe the abstract syntax tree that would be created for the expression `e.c + a[d]`. You can use symbolic names for the offsets.

```plaintext
var a : array [2..5] int
class foo
begin
    var b : int
    var c : int;
end
function bar (d : int)
var e : foo
begin
    print(e.c + a[d])
end
```