Problem 1: Parse Tree and Leftmost Derivation

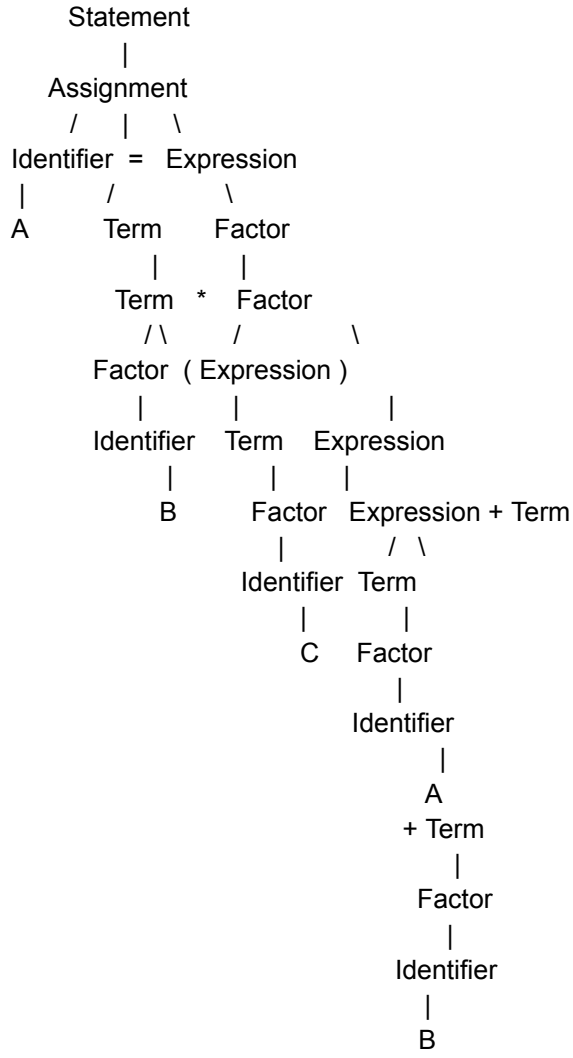
Using the grammar below, show a parse tree and a leftmost derivation for the following statement:

A = B * (C * (A + B))

Grammar:

Statement → Assignment | Expression
Assignment → Identifier = Expression
Expression → Expression + Term | Term
Term → Term * Factor | Factor
Factor → (Expression) | Identifier
Identifier → A | B | C

1.



Problem 2: Scope Concepts

Considering the following program written in pseudocode:

```
int u = 42;
int v = 69;
int w = 17;

proc add( z:int )
  u := v + u + z;

proc bar( fun:proc )
  int u := w;
  fun(v);

proc foo( x:int, w:int )
  int v := x;
  bar(add);

main
  foo(u, 13);
  print(u);
end;
```

a. Using Static Scope, what is printed to the screen?

b. Using Dynamic Scope with Deep Binding, what is printed to the screen?

Hint: The sum for u is 126, but due to deep binding, it's foo's local v that gets involved.

c. Using Dynamic Scope with Shallow Binding, what is printed to the screen?

Hint: The sum for u is 101, but again it's foo's local v that matters.

2.

A. Global scope:

The variables {u = 42}, {v = 69}, and {w = 17} are defined in the global scope at the start of the program. Next, since `u = 42} globally and w = 13, the function foo(u, 13) is called with x = 42. A local variable named v is declared and initialized to x = 42 inside the foo function; hence, v = 42 is within the local scope of foo. bar(add) is then invoked from within foo after that. The global variable w = 17 is assigned to the local variable u in the bar function, meaning that u = 17 in the local scope of bar. Add(v) is then carried out by using the function fun(v). The value of v from bar is supplied to the parameter z in the add(z: int) method. Add obtains z = 69 since bar uses the global v = 69 instead of its local v. The statement u:= v + u + z, where the global values u = 42, v = 69, and z = 69 are from the call to add, is executed inside the add function. It works out to be u = 69 + 42 + 69 = 180. The global u is updated to u = 180 as a result. Ultimately, print(u) publishes the updated value of 180 for the global variable u when it runs. Consequently, **180 is the program's final output**.

B. Deep Binding:

X is set to 42 and w is set to 13 when foo(u, 13) is invoked. A local variable called v is declared and set to x = 42 inside the foo code. Then, bar(add) is called inside foo. A local variable called "u" is declared in bar and assigned the value w = 17. Thus, u = 17 inside the bar. After that, a call is made to the function fun(v), which is actually add(v). The value of v that was transferred from bar to z is assigned in add(z: int). The add function, however, uses the environment in which it was declared because of deep binding, which means that it will reference the variables from the global scope at that particular moment. As a result, the globals u = 42 and v = 69 are utilized by add. The expression u := v + u + z is carried out inside add. In this case, the global values are u = 42, v = 69, and z = 69. The final result is u = 69 + 42 + 69 = 180. The global u is updated to u = 180 as a result. Ultimately, the **global u = 180 is printed to the screen when print(u) is run.**

C. Shallow Binding

x is set to 42 and w to 13 when foo(u, 13) is called. A local variable v is declared inside foo and set to x = 42. Then, from within foo, bar(add) is called. A local variable called u is declared in bar and set to the global value w = 17. In bar, u = 17, so. Next, the call is made to the function fun(v), which is actually add(v). Add will use the environment inside the bar, which is where it is called, in shallow binding. Add this makes use of the u and v from the bar's scope. The expression u = v + u + z is carried out inside add. In this case, z = 42 , v = 42 (the local v from foo), and u = 17 (the local u in bar). This simplifies to u = 42 + 17 + 42 = 101. The global u remains unchanged, but the local u in bar is updated to u = 101. **In the end, print(u) publishes the global u, which remains 42 because it was never updated.**