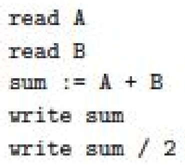
Problem IV. In C, C++ or Python:
(i) Implement the LL(1) parsing algorithm for the PLP calculator language of sections 2.3.1-2.3.3 (2.3.1-2.3.2 in 3rd edition). Run it on example 2.24. Demonstrate with pre-order and post-order listings that your tree is the same as in Figure 2.18 (Figure 2.17 in 3rd edition).

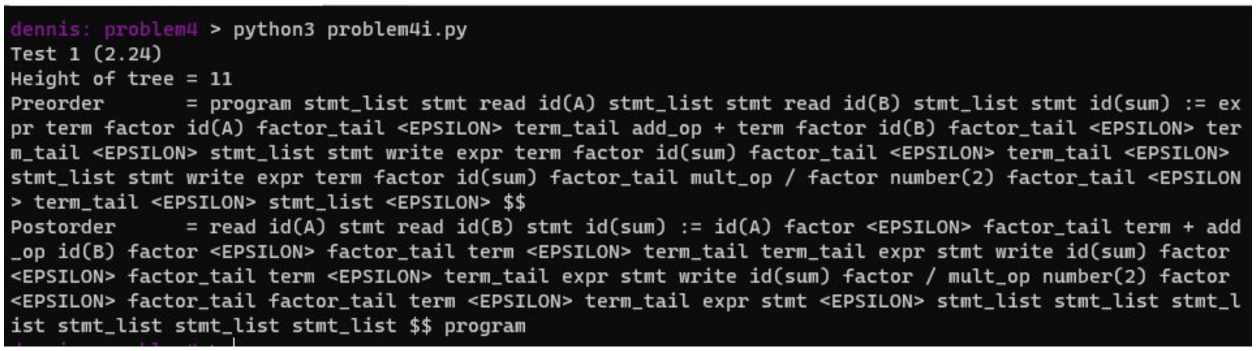
All programming targets were achieved to the 3rd degree. Below is proof of program correctness.

Example 2.24:



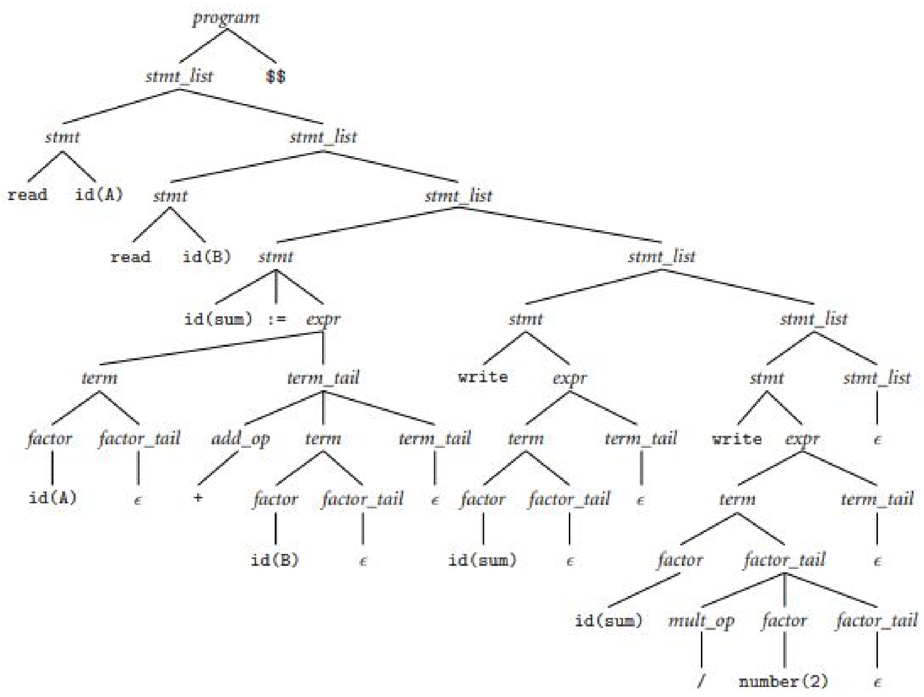
```
read A
read B
sum := A + B
write sum
write sum / 2
```

Below is a screenshot of the LL(1) parsing algorithm running on example 2.24.



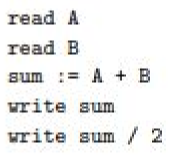
```
dennis: problem4 > python3 problem4i.py
Test 1 (2.24)
Height of tree = 11
Preorder       = program stmt_list stmt read id(A) stmt_list stmt read id(B) stmt_list stmt id(sum) := ex
pr term factor id(A) factor_tail <EPSILON> term_tail add_op + term factor id(B) factor_tail <EPSILON> ter
m_tail <EPSILON> stmt_list stmt write expr term factor id(sum) factor_tail <EPSILON> term_tail <EPSILON>
stmt_list stmt write expr term factor id(sum) factor_tail mult_op / factor number(2) factor_tail <EPSILON
> term_tail <EPSILON> stmt_list <EPSILON> $$
Postorder      = read id(A) stmt read id(B) stmt id(sum) := id(A) factor <EPSILON> factor_tail term + add
_op id(B) factor <EPSILON> factor_tail term <EPSILON> term_tail term_tail expr stmt write id(sum) factor
<EPSILON> factor_tail term <EPSILON> term_tail expr stmt write id(sum) factor / mult_op number(2) factor
<EPSILON> factor_tail factor_tail term <EPSILON> term_tail expr stmt <EPSILON> stmt_list stmt_list stmt_l
ist stmt_list stmt_list stmt_list $$ program
```

The pre/post order that is outputted matches the tree that is shown in Figure 2.18:



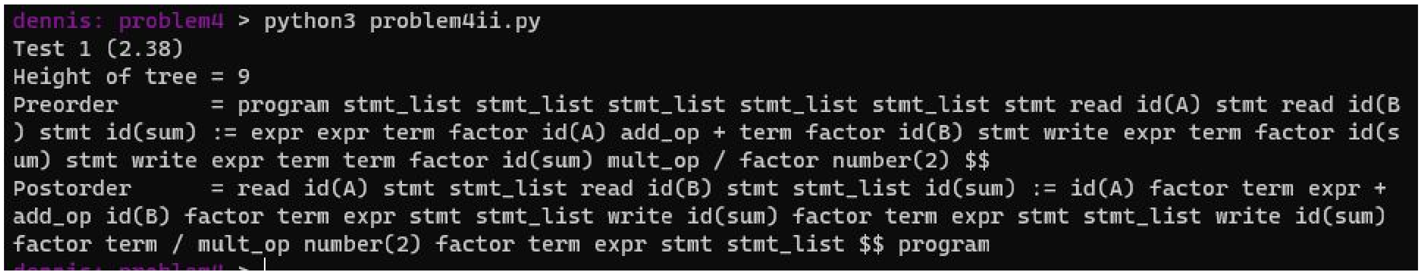
(ii) Implement the LR(1) parsing algorithm for the PLP calculator language of section 2.3.4 (2.3.3 in 3rd edition). Run it on example 2.38. Demonstrate with the pre-order and post-order listings that your parse tree is correct.

Example 2.38



```
read A
read B
sum := A + B
write sum
write sum / 2
```

Below is a screenshot of the LL(1) parsing algorithm running on example 2.38.



```
dennis: problem4 > python3 problem4ii.py
Test 1 (2.38)
Height of tree = 9
Preorder       = program stmt_list stmt_list stmt_list stmt_list stmt_list stmt read id(A) stmt read id(B
) stmt id(sum) := expr expr term factor id(A) add_op + term factor id(B) stmt write expr term factor id(s
um) stmt write expr term term factor id(sum) mult_op / factor number(2) $$
Postorder      = read id(A) stmt stmt_list read id(B) stmt stmt_list id(sum) := id(A) factor term expr +
add_op id(B) factor term expr stmt stmt_list write id(sum) factor term expr stmt stmt_list write id(sum)
factor term / mult_op number(2) factor term expr stmt stmt_list $$ program
dennis: problem4 >
```

The pre/post order that is outputted clearly displays that the generated parse tree is correct.