Contributor: Stephen Hansen Degree: 3

Problem 3. Implement the table driven parsing algorithm of FCS section 11.7 constructing parse trees of the grammar of simple programming statements (FCS figure 11.33). Your algorithm should make use of a stack as explained in FCS figure 11.34. After constructing a parse tree your algorithm should compute its height and list all labels in pre-order and post-order. Demonstrate with examples that your code operates properly.

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

The following examples were used in order to test the correctness of our solution.

Example 11.12. Let us use the parsing table of Fig. 11.32 on the input

```
{w c s ; s ; }ENDM
```

11.7.1: Simulate the table-driven parser using the parsing table of Fig. 11.32 on the following input strings:

a) `{s;}`
b) `wc{s;s;}`
c) `{{s;s;}s;}`
d) `{s;s}`

	(	)	ENDM
<*B*>	2	1	1

Fig. 11.31. Parsing table for the balanced parentheses grammar.

The following code was then developed to solve these examples.

```
TABLE = {
        '<S>': {'w': 1, 'c': None, '{': 2, '}': None, 's': 3, ';': None, 'ENDM': None},
        '<T>': {'w': 4, 'c': None, '{': 4, '}': 5, 's': 4, ';': None, 'ENDM': None}
}

GRAMMAR = [
        ('<S>', 'w c <S>'),
        ('<S>', '{ <T>'),
        ('<S>', 's ;'),
        ('<T>', '<S> <T>'),
        ('<T>', '}')
]
```

```
# Example 11.36 from FCS
print("Test 1 (11.36)")
tree = tableDrivenParser('{ w c s ; s ; } ENDM', TABLE, GRAMMAR)
print("Height of tree = %d" % computeHeight(tree))
print("Preorder       = %s" % preorder(tree))
print("Postorder      = %s" % postorder(tree))

# Example 11.7.1a from FCS
print("Test 2 (11.7.1a)")
tree = tableDrivenParser('{ s ; } ENDM', TABLE, GRAMMAR)
print("Height of tree = %d" % computeHeight(tree))
print("Preorder       = %s" % preorder(tree))
print("Postorder      = %s" % postorder(tree))

# Example 11.7.1b from FCS
print("Test 3 (11.7.1b)")
tree = tableDrivenParser('w c { s ; s ; } ENDM', TABLE, GRAMMAR)
print("Height of tree = %d" % computeHeight(tree))
print("Preorder       = %s" % preorder(tree))
print("Postorder      = %s" % postorder(tree))

# Example 11.7.1c from FCS
print("Test 4 (11.7.1c)")
tree = tableDrivenParser('{ { s ; s ; } s ; } ENDM', TABLE, GRAMMAR)
print("Height of tree = %d" % computeHeight(tree))
print("Preorder       = %s" % preorder(tree))
print("Postorder      = %s" % postorder(tree))

# Example 11.7.1d from FCS. This is an invalid input, intended to fail.
print("Test 5 (11.7.1d)")
tree = tableDrivenParser('{ s ; s } ENDM', TABLE, GRAMMAR)

PARENTHESES_TABLE = {
        '<B>': { '(': 2, ')': 1, 'ENDM': 1 }
}

PARENTHESES_GRAMMAR = [
        ('<B>', ''),
        ('<B>', '( <B> ) <B>')
]

# Simulate on 11.31 table
print("Test 6 (11.31)")
tree = tableDrivenParser('( ( ) ( ) ) ( ( ) ) ENDM', PARENTHESES_TABLE, PARENTHESES_GRAMMAR)
print("Height of tree = %d" % computeHeight(tree))
print("Preorder       = %s" % preorder(tree))
print("Postorder      = %s" % postorder(tree))
```

This is the run output from the above program.

```
Test 1 (11.36)
Height of tree = 4
Preorder       = <S> { <T> <S> w c <S> s ; <T> <S> s ; <T> }
Postorder      = { w c s ; <S> <S> s ; <S> } <T> <T> <T> <S>
Test 2 (11.7.1a)
Height of tree = 3
Preorder       = <S> { <T> <S> s ; <T> }
Postorder      = { s ; <S> } <T> <T> <S>
Test 3 (11.7.1b)
Height of tree = 5
Preorder       = <S> w c <S> { <T> <S> s ; <T> <S> s ; <T> }
Postorder      = w c { s ; <S> s ; <S> } <T> <T> <T> <S> <S>
Test 4 (11.7.1c)
Height of tree = 6
Preorder       = <S> { <T> <S> { <T> <S> s ; <T> <S> s ; <T> } <T> <S> s ; <T> }
Postorder      = { { s ; <S> s ; <S> } <T> <T> <T> <S> s ; <S> } <T> <T> <T> <S>
Test 5 (11.7.1d)
Error! Top of stack does not match lookahead symbol.
Test 6 (11.31)
Height of tree = 3
Preorder       = <B> ( <B> ( <B> ) <B> ( <B> ) <B> ) <B> ( <B> ( <B> ) <B> ) <B>
Postorder      = ( ( <B> ) ( <B> ) <B> <B> <B> ) ( ( <B> ) <B> <B> ) <B> <B> <B>
```

FCS gives the parse tree solution for Test 1:

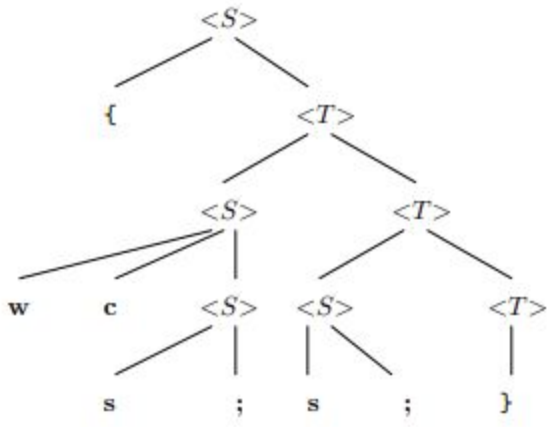

Fig. 11.36. Complete parse tree for the parse of Fig. 11.34.

Comparing our Pre/post order traversals with the image we can see that Test1 was successful.

By manually building and tracing the trees for the rest of the examples, we were able to confirm that the program was working as expected.