A postfix “language” for lab 3

We can combine the rules for a postfix expression with a rule for declaring or initializing a variable to create a simple calculator language. This language is similar to the one in the textbook, but a little simpler because of the use of postfix rather than infix expressions.

prog → stmt | stmt prog
stmt → id = num ; | expr ;
expr → id | num | expr expr op
op → + | - | * | /

It turns out that the bottom-up parse table is as follows. In the reduce entries, I’ve abbreviated the names of the nonterminals to just their first letter.

State	id	=	num	;	+	–	*	/	$	prog	stmt	expr	op
0	2		4								1	3	
1	2		4						☺	5	1	3	
2	-1,E	6	-1,E	-1,E	-1,E	-1,E	-1,E	-1,E					
3	2		4	7								8	
4	-1,E		-1,E	-1,E	-1,E	-1,E	-1,E	-1,E					
5									☺				
6			9										
7	-2,S		-2,S						-2,S				
8	2		4		11	12	13	14				8	10
9				15									
10	-3,E		-3,E	-3,E	-3,E	-3,E	-3,E	-3,E					
11	-1,O		-1,O	-1,O	-1,O	-1,O	-1,O	-1,O					
12	-1,O		-1,O	-1,O	-1,O	-1,O	-1,O	-1,O					
13	-1,O		-1,O	-1,O	-1,O	-1,O	-1,O	-1,O					
14	-1,O		-1,O	-1,O	-1,O	-1,O	-1,O	-1,O					
15	-4,S		-4,S						-4,S				

As you can see, states 2, 4, 7, 10-15 contain reduce actions. Plus, states 1 and 5 are accept states, so these are reductions as well for the end of input. What semantic actions will we need to help us create a syntax tree? We probably won’t be sure until we actually trace through some input to see what we’d expect.

Let’s parse the input: **a = 7 ; 9 a * ;**

State stack	Input	Action	Semantic action we expect

Let’s write out semantic actions here. Can you tell how to figure out which production is being reduced? Do you think we may need some semantic actions for goto operations?

state	Production being reduced	Create what?
1		
2		
4		
5		
7		
10		
11		
12		
13		
14		
15		