Concepts of Programming Languages, CSCI 305, Fall 2021
Homework #8, complete by Nov. 12

The driver for a table-driven SLR(1) parser, along with its parsing table, are given.

```
state = 1 . . number_of_states
symbol = 1 . . number_of_symbols
production = 1 . . number_of_productions
action_rec = record
    action : (shift, reduce, shift_reduce, error)
    new_state : state
    prod : production

parse_tab : array [symbol, state] of action_rec
prod_tab : array [production] of record
    lhs : symbol
    rhs_len : integer
-- these two tables are created by a parser generator tool

parse_stack : stack of record
    sym : symbol
    st : state
parse_stack.push(⟨null, start_state⟩)
cur_sym : symbol := scan                    -- get new token from scanner
loop
    cur_state : state := parse_stack.top.st -- peek at state at top of stack
    if cur_state = start_state and cur_sym = start_symbol
        return                              -- success!
    ar : action_rec := parse_tab[cur_state, cur_sym]
    case ar.action
        shift:
            parse_stack.push(⟨cur_sym, ar.new_state⟩)
            cur_sym := scan                 -- get new token from scanner
        reduce:
            cur_sym := prod_tab[ar.prod].lhs
            parse_stack.pop(prod_tab[ar.prod].rhs_len)
        shift_reduce:
            cur_sym := prod_tab[ar.prod].lhs
            parse_stack.pop(prod_tab[ar.prod].rhs_len-1)
        error:
            parse_error
```

Figure 2.28 **Driver for a table-driven SLR(1) parser.** We call the scanner directly, rather than using the global input_token of Figures 2.16 and 2.18, so that we can set cur_sym to be an arbitrary symbol.

Top-of-stack state	Current input symbol																		
	sl	*s*	*e*	*t*	*f*	*ao*	*mo*	id	lit	r	w	:=	(	)	+	-	*	/	$$
0	s2	b3	–	–	–	–	–	s3	–	s1	s4	–	–	–	–	–	–	–	–
1	–	–	–	–	–	–	–	b5	–	–	–	–	–	–	–	–	–	–	–
2	–	b2	–	–	–	–	–	s3	–	s1	s4	–	–	–	–	–	–	–	b1
3	–	–	–	–	–	–	–	–	–	–	–	s5	–	–	–	–	–	–	–
4	–	–	s6	s7	b9	–	–	b12	b13	–	–	–	s8	–	–	–	–	–	–
5	–	–	s9	s7	b9	–	–	b12	b13	–	–	–	s8	–	–	–	–	–	–
6	–	–	–	–	–	s10	–	r6	–	r6	r6	–	–	–	b14	b15	–	–	r6
7	–	–	–	–	–	–	s11	r7	–	r7	r7	–	–	r7	r7	r7	b16	b17	r7
8	–	–	s12	s7	b9	–	–	b12	b13	–	–	–	s8	–	–	–	–	–	–
9	–	–	–	–	–	s10	–	r4	–	r4	r4	–	–	–	b14	b15	–	–	r4
10	–	–	–	s13	b9	–	–	b12	b13	–	–	–	s8	–	–	–	–	–	–
11	–	–	–	–	b10	–	–	b12	b13	–	–	–	s8	–	–	–	–	–	–
12	–	–	–	–	–	s10	–	–	–	–	–	–	–	b11	b14	b15	–	–	–
13	–	–	–	–	–	–	s11	r8	–	r8	r8	–	–	r8	r8	r8	b16	b17	r8

Figure 2.27 SLR(1) parse table for the calculator language. Table entries indicate whether to shift (s), reduce (r), or shift and then reduce (b). The accompanying number is the new state when shifting, or the production that has been recognized when (shifting and) reducing. Production numbers are given in Figure 2.24. Symbol names have been abbreviated for the sake of formatting. A dash indicates an error. An auxiliary table, not shown here, gives the left-hand-side symbol and right-hand-side length for each production.

Grammar:

1. program → stmt_list $$
2. stmt_list → stmt_list stmt
3. stmt_list → stmt
4. stmt → id := expr
5. stmt → read id
6. stmt → write expr
7. expr → term
8. expr → expr add_op term
9. term → factor
10. term → term mult_op factor
11. factor → (expr)
12. factor → id
13. factor → number
14. add_op → +
15. add_op → -
16. mult_op → *
17. mult_op → /

Hand execute the code on the program:

sum:=A+B $$

showing all table lookups, the input stream and the stack.

table lookup	stack (sym, state) cur_state is the top of the stack	cur_sym	input stream
			sum:=A+B $$
	(null,0)	id (sum)	:=A+B $$
For each execution of the loop:			
(0, id) → s3	(null, 0)(id, 3)	:=	A+B $$
(3, :=) → s5	(null, 0)(id, 3)(:=, 5)	id(A)	+B $$
(5, id) → b12	(null, 0)(id, 3)(:=, 5)	factor	+B $$
(5, factor) → b9	(null, 0)(id, 3)(:=, 5)	term	+B $$
(5, term) → s7	(null, 0)(id, 3)(:=, 5)(term, 7)	+	B $$
(7, +) → r7	(null, 0)(id, 3)(:=, 5)	expr	+B $$ (reduce puts + back)
(5, expr) → s9	(null, 0)(id, 3)(:=, 5)(expr, 9)	+	B $$
(9, +) → b14	(null, 0)(id, 3)(:=, 5)(expr, 9)	add_op	B $$
(9, add_op) → s10	(null, 0)(id, 3)(:=, 5)(expr,9)(add_op, 10)	id (B)	$$
(10, id) → b12	(null, 0)(id, 3)(:=, 5)(expr,9)(add_op, 10)	factor	$$
(10, factor) → b9	(null, 0)(id, 3)(:=, 5)(expr,9)(add_op, 10)	term	$$
(10, term) → s13	(null, 0)(id, 3)(:=, 5)(expr,9)(add_op, 10)(term, 13)	$$	
(13, $$) → r8	(null, 0)(id, 3)(:=, 5)	expr	$$ (reduce puts $$ back)
(5, expr) → s9	(null, 0)(id, 3)(:=, 5)(expr, 9)	$$	
(9, $$) → r4	(null, 0)	stmt	$$ (reduce puts $$ back)
(0, stmt) → b3	(null, 0)	stmt_list	$$
(0, stmt_list) → s2	(null, 0)(stmt_list, 2)	$$	
(2, $$) → b1	(null, 0)	program	
cur_state=0 & cur_sym=program so return☺			