Assignment 4, LR(1) Parser

Due on Nov. 14, 11:59:59 PM (Wednesday)

Consider the following grammar and its LR(1) parsing table for expressions:

$$\begin{aligned} E &\longrightarrow E + T \mid T \\ T &\longrightarrow T * F \mid F \\ F &\longrightarrow (E) \mid n \qquad \text{where } n \text{ is any positive integer.} \end{aligned}$$

	n	$+$	$*$	(	)	\$	E	T	F
0	shift 5			shift 4			1	2	3
1		shift 6				accept			
2		$E \to T$	shift 7		$E \to T$	$E \to T$			
3		$T \to F$	$T \to F$		$T \to F$	$T \to F$			
4	shift 5			shift 4			8	2	3
5		$F \to$ `id`	$F \to$ `id`		$F \to$ `id`	$F \to$ `id`			
6	shift 5			shift 4				9	3
7	shift 5			shift 4					10
8		shift 6			shift 11				
9		$E \to E + T$	shift 7		$E \to E + T$	$E \to E + T$			
10		$T \to T * F$	$T \to T * F$		$T \to T * F$	$T \to T * F$			
11		$F \to (E)$	$F \to (E)$		$F \to (E)$	$F \to (E)$			

As before, let \$ denote the end of the expression, but the user doesn't have to put \$ at the end of the expression. Your programs should prepare the token string with \$ attached automatically.

Implement an LR(1) parser according to the parsing table above. In addition to indicating whether or not the input expression is valid, whenever the stack is updated (push or pop) your parser should print out the contents of the stack (from bottom of the stack to its top) followed by the remaining tokens string. (See following example where the input is 123+456*789.)

```
  [-:0][n:5]    + n * n $
  [-:0][F:3]    + n * n $
  [-:0][T:2]    + n * n $
  [-:0][E:1]    + n * n $
  [-:0][E:1][+:6]    + n * n $
  [-:0][E:1][+:6][n:5]    * n $
  [-:0][E:1][+:6][F:3]    * n $
  [-:0][E:1][+:6][T:9]    * n $
  [-:0][E:1][+:6][T:9][*:7]    * n $
  [-:0][E:1][+:6][T:9][*:7][n:5]    $
  [-:0][E:1][+:6][T:9][*:7][F:10]    $
  [-:0][E:1][+:6][T:9]    $
  [-:0][E:1]    $
-- > Valid Expression!
```

For example, `[F:3]` is an item in the stack, where `F` is obtained from some reduction and `3` is the state after the reduction determined by the parsing table. If `[F:3]` is at the left-most position (i.e., on the top of the stack), then the current state is 3, and `* n $` follows it is the remaining token string.

Note: Follow the submission guidelines, and make sure it can be compiled and run on Unix. Don't submit your works to a wrong section and project.