Bottom-Up (Shift–Reduce) Parsing for (id + id) * id

1. Grammar

We use the classic expression grammar with precedence and associativity:

E → E + T | T
T → T * F | F
F → (E) | id

2. Input String

We will parse the input string: (id + id) * id
For LR-style shift–reduce parsing we append the end marker $,
so the full input becomes: (id + id) * id $

3. Shift–Reduce Concepts

- Stack: symbols we have already shifted.
- Input: symbols we have not yet consumed.
- **Shift**: move the next input symbol onto the stack.
- **Reduce**: if the top of the stack matches the right-hand side (RHS) of a production, replace that RHS by the left-hand side (LHS) nonterminal.
- Accept: we accept when the stack is just the start symbol E and the remaining input is $.

Note: The stack is shown in the table as bottom→top (left to right).

4. Detailed Shift–Reduce Trace

At each step we either SHIFT the next input symbol,
or REDUCE using one of the productions:

E → E + T
E → T
T → T * F
T → F
F → (E)
F → id

Step	Stack (bottom→top)	Remaining input	Action
0	[ ]	(id + id) * id $	start
1	(	id + id) * id $	shift '('
2	(id	+ id) * id $	shift id
3	(F	+ id) * id $	reduce F → id
4	(T	+ id) * id $	reduce T → F
5	(E	+ id) * id $	reduce E → T
6	(E +	id) * id $	shift '+'
7	(E + id	) * id $	shift id
8	(E + F	) * id $	reduce F → id
9	(E + T	) * id $	reduce T → F
10	(E	) * id $	reduce E → E + T
11	(E)	* id $	shift ')'
12	F	* id $	reduce F → (E)
13	T	* id $	reduce T → F
14	T *	id $	shift '*'
15	T * id	$	shift id
16	T * F	$	reduce F → id
17	T	$	reduce T → T * F
18	E	$	reduce E → T
19	E	$	Accept

5. Interpretation

After step 18, the stack has only E and the input is $, so in step 19 we ACCEPT.
This confirms that (id + id) * id is in the language of the grammar. Bottom-up (shift–reduce) parsing reconstructs the rightmost derivation in reverse, starting from the concrete input tokens and reducing them back to the start symbol E.