Name	Roll No.	Section	DoP
Vidhan Dahatkar	166	B	

Practical - 07

Aim- Design and Simulate SLR(1) parsing using JFLAP for the grammar rules:
E→ E+T | T, T→ T*F |F, F → (E) | id and parse the sentence: id + id * id.

Theory:

SLR(1) Parsing:

SLR(1) (Simple LR(1)) parsing is a type of **bottom-up parsing** that uses **one lookahead symbol** (hence the "1" in SLR(1)) and constructs a **rightmost derivation** in reverse. This technique is suitable for parsing grammars that can be efficiently processed without backtracking. The SLR(1) parser is an extension of the **LR(0)** parser, which uses a **parsing table** to determine which actions to take (shift, reduce, accept, or error) based on the current state and lookahead symbol.

Key components of **SLR(1) parsing**:

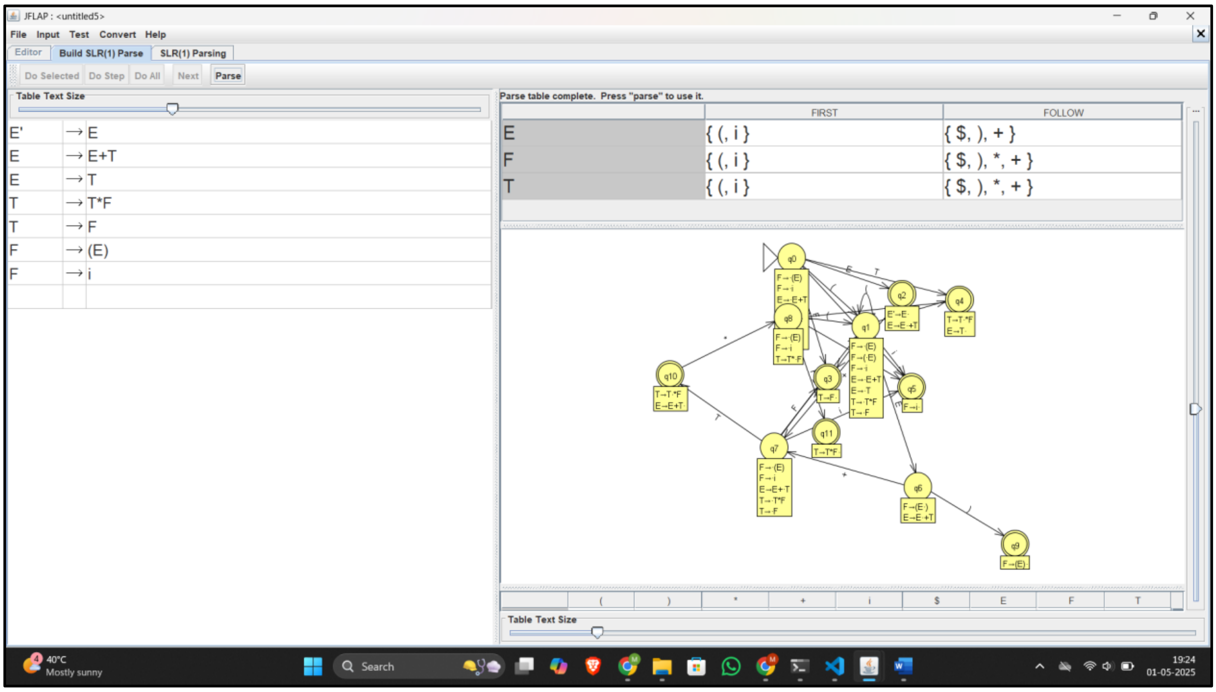
1. **LR(0) Items**:
   - An **LR(0) item** represents a production rule where a dot (.) indicates the current position of the parser. The parser moves the dot to recognize symbols step by step.
   - For example, for the rule E → E + T, the item E → E + T (with the dot at the beginning) means that the parser expects to see E + T to match the non-terminal E.
2. **SLR(1) Parsing Table**:
   - The **SLR(1) parsing table** is a 2D table where rows represent states and columns represent terminal symbols and non-terminal symbols. Each cell contains an action (shift, reduce, accept, or error).
   - **Shift** means moving the next input symbol onto the stack.
   - **Reduce** means replacing a portion of the stack with a non-terminal based on a production rule.
   - **Accept** indicates that the parsing has successfully finished.
   - **Error** indicates that the parsing cannot continue due to a conflict.
3. **First and Follow Sets**:
   - **First Sets**: For a non-terminal, the **First** set contains all possible terminal symbols that can start strings derived from that non-terminal.

- **Follow Sets**: The **Follow** set of a non-terminal contains all the terminal symbols that can appear immediately after that non-terminal in any derivation.

4. **Parsing Process**:

   - The parser starts with an empty stack and the input string to parse. It uses the **SLR(1) parsing table** to determine the appropriate action based on the current state and the lookahead symbol.
   - If the table suggests **shift**, the next input symbol is pushed onto the stack.
   - If the table suggests **reduce**, a portion of the stack is replaced with the non-terminal according to the grammar's production rules.
   - The parser continues this process until it either successfully **accepts** the input or encounters an **error**.

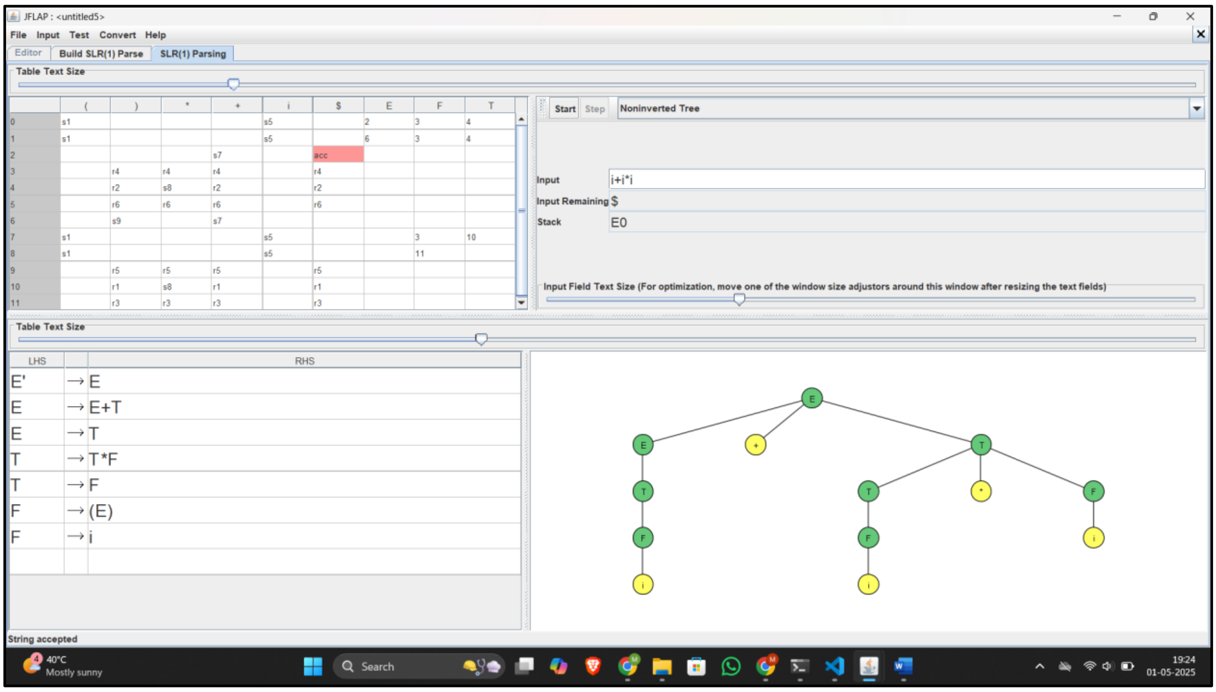
Output:



E' → E
E → E+T
E → T
T → T*F
T → F
F → (E)
F → i

Parse table complete. Press "parse" to use it.

	FIRST	FOLLOW
E	{ (, i }	{ $,), + }
F	{ (, i }	{ $,), *, + }
T	{ (, i }	{ $,), *, + }


JFLAP : <untitled5>
File Input Test Convert Help
Editor
Build SLR(1) Parse
SLR(1) Parsing
Table Text Size
Start
Step
Noninverted Tree
Input Field Text Size (For optimization, move one of the window size adjustors around this window after resizing the text fields)
E
E
+
T
T
T
*
F
F
F
i
i
i
i

	(	)	*	+	i	$	E	F	T
0	s1				s5		2	3	4
1	s1				s5		6	3	4
2				s7		acc			
3		r4	r4	r4		r4			
4		r2	s8	r2		r2			
5		r6	r6	r6		r6			
6		s9		s7					
7	s1				s5			3	10
8	s1				s5			11	
9		r5	r5	r5		r5			
10		r1	s8	r1		r1			
11		r3	r3	r3		r3			

Input i+i*i

Input Remaining $

Stack E0

LHS		RHS
E'	→	E
E	→	E+T
E	→	T
T	→	T*F
T	→	F
F	→	(E)
F	→	i

String accepted

Conclusion:

In this practical, we learned how to design and simulate an **SLR(1) parser** using **JFLAP**. We followed the process of constructing the **SLR(1) parsing table**, which involves calculating the **First** and **Follow** sets for each non-terminal in the grammar. The parsing process was simulated using JFLAP, where we parsed an arithmetic expression id + id * id using the constructed parsing table.