Group Project-Part 3 (SLR Parser)
CPCS-302-CS1, CS2

Assigned on: 25-Apr-2024 **Due Date: 09-May-2024**

CLO: 12	SO: 1

Objective:

- To learn how to implement Simple LR (SLR) parser using LR parse table.
- To enhance programming skills and learn how to work as a team

Student Outcome Covered:

SO # 1: Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions.

Problem Statement:

Consider the following augmented grammar:

T' ⟶ T
1. T ⟶ R
2. T ⟶ a T c
3. R ⟶ Є
4. R ⟶ b R

The SLR parse table for this grammar is as follows:

	Actions				GoTo	
	a	b	c	$	*T*	*R*
0	s3	s4	r3	r3	1	2
1				Accept		
2			r1	r1		
3	s3	s4	r3	r3	5	2
4		s4	r3	r3		6
5			s7			
6			r4	r4		
7			r2	r2		

Write a complete Java program (Using the algorithm given at the end of this description) for constructing SLR parser for the above grammar using the above parse table.

Sample Input and Output:

If the input file (input.txt) **contains the following inputs (tokens are separated with spaces):**

a b b c $

a a b c c $

the output on screen should be as follows:

Right most derivation for the input a b b c**:**

Stack	Input	Action
0	**a b b c $**	S3 (Shift 3)
0 a 3	b b c $	S4 (Shift 4)
0 a 3 b 4	b c $	S4 (Shift 4)
0 a 3 b 4 b 4	c $	R3 (Reduce by R → Є
0 a 3 b 4 b 4 R 6	c $	R4 (Reduce by R → bR
0 a 3 b 4 R 6	c $	R4 (Reduce by R → bR
0 a 3 R 2	c $	R1 (Reduce by T → R
0 a 3 T 5	c $	S7 (Shift 7)
0 a 3 T 5 c 7	$	R2 (Reduce by T → a T c
0 T 1	$	Accept

Right most derivation for the input a a b c c $**:**

Stack	Input	Action
0	a a b c c $	S3 (Shift 3)
0 a 3	a b c c $	S3 (Shift 3)
0 a 3 a 3	b c c $	S4 (Shift 4)
0 a 3 a 3 b 4	c c $	R3 (Reduce by R→ Є
0 a 3 a 3 b 4 R 6	c c $	R4 (Reduce by R→ bR
0 a 3 a 3 R 2	c c $	R1 (Reduce by T→ R
0 a 3 a 3 T 5	c c $	S7 (Shift 7)
0 a 3 a 3 T 5 c 7	c $	R2 (Reduce by T→ aTc
0 a 3 T 5	c $	S7 (Shift 7)
0 a 3 T 5 c 7	$	R2 (Reduce by T→ aTc
0 T 1	$	Accept

SLR Parsing Algorithm

SLR parsing algorithm.

Input. An input string w and an LR parsing table with functions *action* and *goto* for a grammar G.

Output. If w is in $L(G)$, a bottom-up parse for w; otherwise, an error indication.

Method. Initially, the parser has s_0 on its stack, where s_0 is the initial state, and $w\$$ in the input buffer. The parser then executes the program until an accept or error action is encountered.

```
set ip to point to the first symbol of w$;
repeat forever begin
    let s be the state on top of the stack and
        a the symbol pointed to by ip;
    if action[s, a] = shift s' then begin
        push a then s' on top of the stack;
        advance ip to the next input symbol
    end
    else if action[s, a] = reduce A → β then begin
        pop 2*|β| symbols off the stack;
        let s' be the state now on top of the stack;
        push A then goto[s', A] on top of the stack;
        output the production A → β
    end
    else if action[s, a] = accept then
        return
    else error()
end
```

Restrictions:

(i) You should use Hash Table or Two dimensional array to store above parse table.

(ii) You should **read all inputs** from the file **"input.txt"** and generate the **outputs** for **all** these inputs **on screen.**

(iii) As soon as parser finds some syntax error, it should display the error message "Syntax Error" and exit the program.

Important Note:

- Every group should consist of maximum 3 students of your own choice.
- Any kind of plagiarism/cheating will result in 0 marks.
- No late submission will be accepted.
- Only one student of each group should upload the solution (compressed files: .zip, .7z or .rar containing java files) on Blackboard on or before due date.
- Write ID, Name and Email of each member of the group as comment in each Java file.
- No solution will be accepted through email.
- Solution in any other file (.pdf, .docx, etc.) format will not be entertained and will result in 0 marks.
- Your program should generate output exactly similar to the above sample outputs.