Name: Saloni Vishwakarma

Roll No. 13

Practical No. 4

Topic: Parser Construction

Platform: Windows or Linux

Language to be used: C, Python or Java (Choose based on the companies targeted for placement)

Aim: Implement SLR Parser.

Theory:

An SLR (Simple LR) parser is a type of bottom-up parser commonly used in compiler construction to analyze the syntax of a programming language. It operates by shifting input symbols onto a stack until it can reduce them to grammar productions, ultimately recognizing whether the input string conforms to the grammar rules of the language.

Here's a simplified overview of how an SLR parser works:

1. Input: The parser takes an input string (sequence of tokens) from the lexical analyser.

2. Stack: It maintains a stack to store grammar symbols and parser states.

3. Parsing Table: The SLR parser uses a parsing table constructed from the grammar to determine its actions. This table typically consists of rows representing states and columns representing input symbols. The entries in the table specify whether to shift, reduce, or accept based on the current state and the next input symbol.

4. Shift Operation: If the parsing table indicates a "shift" operation for the current state and input symbol, the parser shifts the input symbol onto the stack and transitions to a new state.

5. Reduce Operation: If the parsing table indicates a "reduce" operation, the parser pops grammar symbols from the stack according to the right-hand side of a production rule and replaces them with the corresponding non-terminal symbol. This step aims to recognize higher-level constructs in the input.

6. Acceptance: If the parsing table indicates "accept" for a certain state and the end-of-input symbol, the parser successfully recognizes the input string as valid according to the grammar.

SLR parsers are relatively simple and efficient, but they require a grammar that adheres to certain restrictions, such as being in the SLR(1) grammar class. While SLR parsers may not handle all grammars efficiently, they serve as educational tools for understanding parsing algorithms and are often used in introductory compiler courses.

A. Construct Shift-Reduce Parser

C, Python or Java (Choose based on the companies targeted for placement)

Odd Roll number

S –> S + S

S –> S * S

S –> id

Perform Shift Reduce parsing for input string “id + id + id”.

Stack	Input Buffer	Parsing Action
$	id+id+id$	Shift
$id	+id+id$	Reduce S->id
$S	+id+id$	Shift
$S+	id+id$	Shift
$S+id	+id$	Reduce S->id
$S+S	+id$	Reduce S->S+S
$S	+id$	Shift
$S+	id$	Shift
$S+id	$	Reduce S->id
$S+S	$	Reduce S->S+S
$S	$	Accept

Even Roll number

E –> 2E2

E –> 3E3

E –> 4

Perform Shift Reduce parsing for input string “32423”.

Stack	Input Buffer	Parsing Action
$	32423$	Shift
$3	2423$	Shift
$32	423$	Shift
$324	23$	Reduce by E --> 4
$32E	23$	Shift
$32E2	3$	Reduce by E --> 2E2
$3E	3$	Shift
$3E3	$	Reduce by E --> 3E3
$E	$	Accept

Code

```
#include <iostream>
#include <cstring>
using namespace std;

int i = 0, j = 0, c = 0;

char a[16], stk[15];

void check()
{
   for (int z = 0; z < c; z++)
   {
      if (stk[z] == '@')
      {
         cout << " REDUCE TO S -> @" << endl;
         stk[z] = 'S';
         stk[z + 1] = '\0';
         cout << "$" << stk << "\t" << a << "$\t";
      }
   }

   for (int z = 0; z < c - 2; z++)
   {
      if (stk[z] == 'S' && stk[z + 1] == '+' &&
         stk[z + 2] == 'S')
      {
         cout << " REDUCE TO S -> S+S" << endl;
         stk[z] = 'S';
         stk[z + 1] = '\0';
         stk[z + 2] = '\0';
         cout << "$" << stk << "\t" << a << "$\t";
         i = i - 2;
      }
   }

   for (int z = 0; z < c - 2; z++)
   {
      if (stk[z] == 'S' && stk[z + 1] == '*' &&
         stk[z + 2] == 'S')
      {
         cout << " REDUCE TO S -> S*S" << endl;
         stk[z] = 'S';
         stk[z + 1] = '\0';
         stk[z + 2] = '\0';
         cout << "$" << stk << "\t" << a << "$\t";
         i = i - 2;
      }
   }
   return;
}
```

```
int main()
{
   cout << "GRAMMAR is -\nS->S+S \nS->S*S \nS->@\n";

   strcpy(a, "@+@+@");

   c = strlen(a);

   cout << "\nstack \t input \t action";

   cout << "\n$\t" << a << "$\t";

   for (i = 0; j < c; i++, j++)
   {
      stk[i] = a[j];
      stk[i + 1] = '\0';

      a[j] = ' ';

      cout << " SHIFT\n";
      cout << "$" << stk << "\t" << a << "$\t";

      check();
   }

   check();

   if (stk[0] == 'S' && stk[1] == '\0')
      cout << " Accept\n";
   else
      cout << " Reject\n";

   return 0;
}
```

Output Screenshot