**17. Design an LALR Bottom-Up Parser for the given grammar**

**AIM:** To design and implement an LALR bottom-up parser for checking the syntax of the statements in the language.

**THEORY:** LALR Parser is a lookahead LR parser. It is the most powerful parser which can handle large classes of grammar. LALR works similarly to CLR. The only difference is, it combines the similar states of the CLR parsing table into one single state.

**LALR Bottom-up Parser:**

<parser.l>

%{

#include<stdio.h>

#include "y.tab.h"

%}

%%

[0-9]+ {yylval.dval=atof(yytext);

return DIGIT;

}

\n|. return yytext[0];

%%

<parser.y>

%{

/\*This YACC specification file generates the LALR parser for the program

considered in experiment 4.\*/

#include<stdio.h>

%}

%union

{

double dval;

}

%token <dval> DIGIT

%type <dval> expr

%type <dval> term

%type <dval> factor

%%

line: expr '\n' {

;

printf("%g\n",$1);

}

expr: expr '+' term {$$=$1 + $3 ;}

| term

;

term: term '\*' factor {$$=$1 \* $3 ;}

| factor

;

factor: '(' expr ')' {$$=$2 ;}

| DIGIT

;

%%

int main()

{

yyparse();

}

yyerror(char \*s)

{

printf("%s",s);

}

**OUTPUT:**

$lex parser.l

$yacc –d parser.y

$cc lex.yy.c y.tab.c –ll –lm

$./a.out

2+3

5.0000