Compiler Design Lab

Assignment - 5

Syntax Analysis/Parsing- (LL(1) predictive parser)

Name:- T.V.S.S.Sripad

Roll Number:- 18CS01008

Instructions to run program:-

1. To run program for 1st grammer run:-
    - a) Enter input to be parsed in input1.txt file
    - b) ./run.sh 1
2. To run program with 2nd grammar:-
    - 1) Enter input to be parsed in input2.txt file
    - b) ./run.sh 2

Note:-

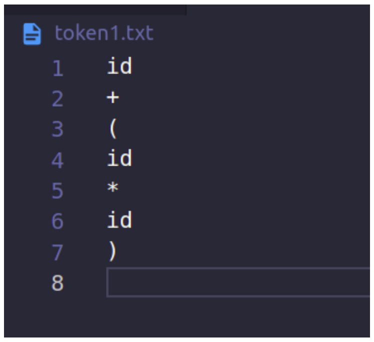
1) Grammar1 refers to the following grammar:-
   E -> E + T | T
   T -> T*F | F
   F -> (E) | id
2) Grammar2 refers to the following grammar:-
   P → prog DL SL end
   DL → D DL | ε
   D → TY VL ;
   TY → int | float
   VL → id VL | id
   SL → S SL | ε
   S → ES | IS | WS | IOS
   ES → id := E ;
   IS → if BE then SL end |
   if BE then SL else SL end
   WS → while BE do SL end
   IOS → print PE | scan id
   PE → E | str
   BE → BE or AE | AE
   AE → AE and NE | NE
   NE → not NE | { BE } | RE
   RE → E = E | E < E | E > E
   E → E + T | E − T | T
   T → T ∗ F | T / F | F
   F → (E) | id | ic | fc

3) The left-factored grammars for the given 1st grammar, 2nd grammar are stored in grammar1.txt, grammar2.txt files respectively.
4) The input to be parsed is to be given in input1.txt, input2.txt for grammar1, grammar2 respectively.
5) First sets, Follow sets for the given grammar are stored in firstsFollows.py file in the form of a python-dictionary data-structure.
6) lexan1.l, lexan2.l are the files with lex codes for tokenizing the given input according to grammar1 and grammar2 respectively.
7) token1.txt, token2.txt are intermediate files generated after lexical analysis. These files contain tokens identified from given input.
8) main.py is the main file which contains code for predictive parsing.
9) parseTable.py file has code for generating parseTable. It takes grammar, first set, follow set, terminal symbols as parameters. This file is used by main.py file to generate parseTable before parsing.
10) run.sh file takes grammar number (question number 1 or 2) as input and performs lexical analysis followed by parsing and writes the final output to finalOutput.txt file.
11) String '_eps' has been used to denote 'Epsilon' everywhere.
12) Input format for grammar in grammar1.txt or grammar2.txt is as follows:
    a) First line consists of set of comma separated non-terminals
    b) Second line consists of set of comma separated terminals
    c) Then production rules follow. The production A -> BCD | E is to be given as:-
       A -> B C D | E (Note that B,C,D are to be separated by space)

Working:-
Run.sh file takes question number as parameter and calls corresponding lexan1.l or lexan2.l files which take input from input1.txt, input2.txt respectively. Then the corresponding lex.yy.c file is compiled and executed.
The output after lexical analysis i.e., tokens detected from input are written to token1.txt or token2.txt depending on the question number. Then the main.py file is called for parsing.
The main.py file reads grammar from grammar1.txt or grammar2.txt and also reads first sets, follow sets from firstsFollows.py and calls parseTable.py file for building parseTable.
After the parseTable is built, the main.py file reads tokens which were written during lexical analysis and with the help of parseTable performs predictive-parsing and output is printed on terminal and is also written to finalOutput.txt file

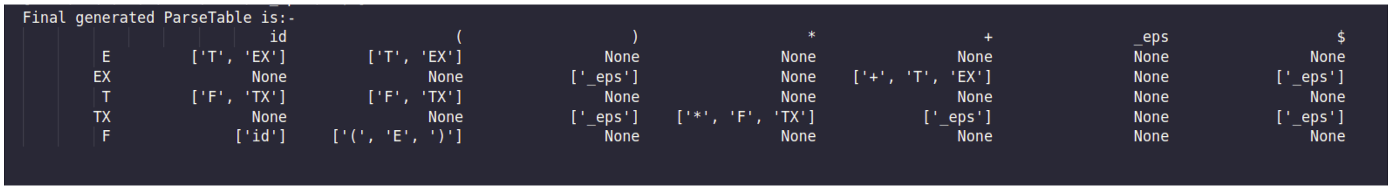
Sample Examples:-
1. For grammar1, **x + (y*z)** is given as input. The output for this execution can also be found in example1.txt file.
   Input : x + (y*z)

Output from lexical analysis:-



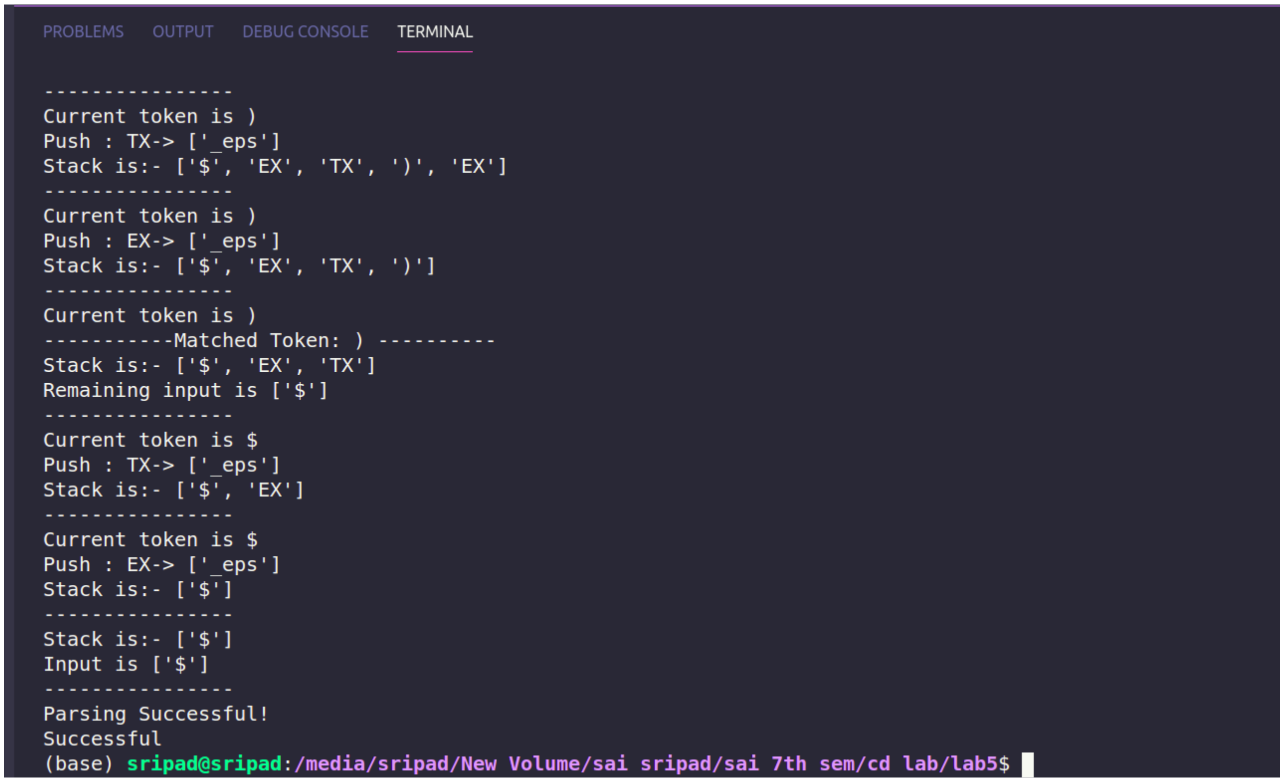
```
token1.txt
id
+
(
id
*
id
)

```

Generated parseTable for grammar1 is



```
Final generated ParseTable is:-
                    id               (               )               *               +            _eps               $
      E     ['T', 'EX']     ['T', 'EX']            None            None            None            None            None
     EX            None            None       ['_eps']            None ['+', 'T', 'EX']            None        ['_eps']
      T     ['F', 'TX']     ['F', 'TX']            None            None            None            None            None
     TX            None            None       ['_eps'] ['*', 'F', 'TX']        ['_eps']            None        ['_eps']
      F         ['id']  ['(', 'E', ')']            None            None            None            None            None
```

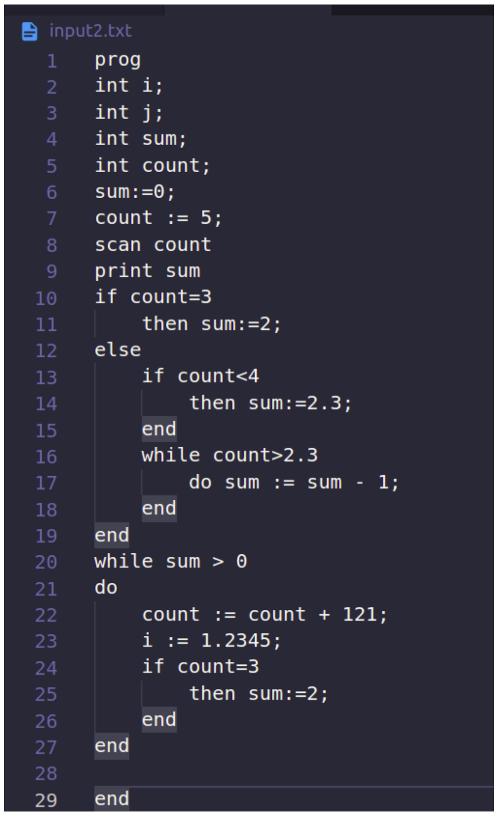
Last lines of output for given example:- (Complete output is attached in example1.txt file)



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
----------------
Current token is )
Push : TX-> ['_eps']
Stack is:- ['$', 'EX', 'TX', ')', 'EX']
----------------
Current token is )
Push : EX-> ['_eps']
Stack is:- ['$', 'EX', 'TX', ')']
----------------
Current token is )
-----------Matched Token: ) ----------
Stack is:- ['$', 'EX', 'TX']
Remaining input is ['$']
----------------
Current token is $
Push : TX-> ['_eps']
Stack is:- ['$', 'EX']
----------------
Current token is $
Push : EX-> ['_eps']
Stack is:- ['$']
----------------
Stack is:- ['$']
Input is ['$']
----------------
Parsing Successful!
Successful
(base) sripad@sripad:/media/sripad/New Volume/sai sripad/sai 7th sem/cd lab/lab5$ 
```

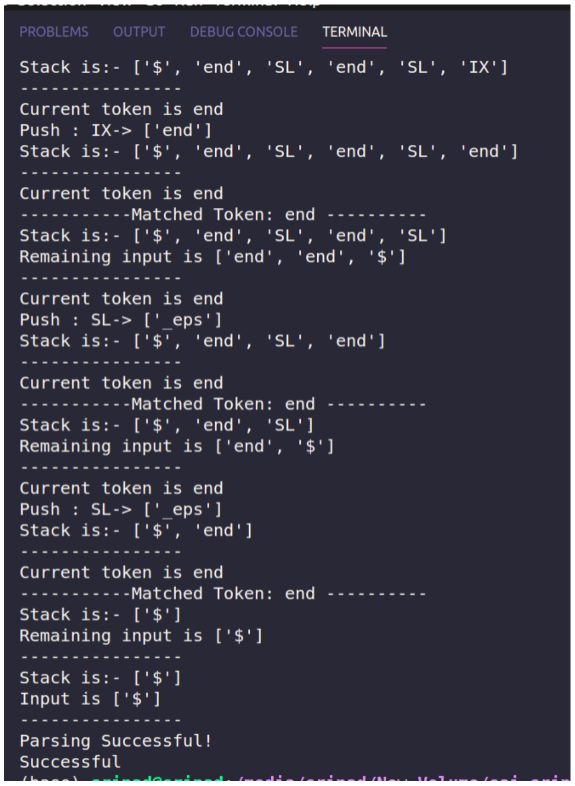
Example2:



input2.txt

```
prog
int i;
int j;
int sum;
int count;
sum:=0;
count := 5;
scan count
print sum
if count=3
    then sum:=2;
else
    if count<4
        then sum:=2.3;
    end
    while count>2.3
        do sum := sum - 1;
    end
end
while sum > 0
do
    count := count + 121;
    i := 1.2345;
    if count=3
        then sum:=2;
    end
end

end
```

Last lines of execution for given input (Complete output has been attached in example2.txt file)



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Stack is:- ['$', 'end', 'SL', 'end', 'SL', 'IX']
----------------
Current token is end
Push : IX-> ['end']
Stack is:- ['$', 'end', 'SL', 'end', 'SL', 'end']
----------------
Current token is end
----------Matched Token: end ----------
Stack is:- ['$', 'end', 'SL', 'end', 'SL']
Remaining input is ['end', 'end', '$']
----------------
Current token is end
Push : SL-> ['_eps']
Stack is:- ['$', 'end', 'SL', 'end']
----------------
Current token is end
----------Matched Token: end ----------
Stack is:- ['$', 'end', 'SL']
Remaining input is ['end', '$']
----------------
Current token is end
Push : SL-> ['_eps']
Stack is:- ['$', 'end']
----------------
Current token is end
----------Matched Token: end ----------
Stack is:- ['$']
Remaining input is ['$']
----------------
Stack is:- ['$']
Input is ['$']
----------------
Parsing Successful!
Successful
```