Name: EKKULURI RAJESH

Registration Number: RA2011003010669

Mail ID: er4359@srmist.edu.in

Department: COMPUTING TECHNOLOGIES

Specialization: CORE

Semester: VI

Subject Title: Compiler Design

Handled By: Dr.G.Abirami

18CSC304J

COMPILER DESIGN

COMMAND LINE CALCULATOR

MINOR PROJECT REPORT

Submitted by

G. Tharun (RA2011003010660)

A.Chandra Mouli(RA2011003010664)

E. Rajesh (RA2011003010669)

Under the guidance of

Dr.G.ABIRAMI

for the course

18CSC304J-Compiler Design

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report "**Command Line Calculator**" is the bonafide work of G. Tharun(RA2011003010660),A.Chandra Mouli(RA2011003010664)and E. Rajesh(RA2011003010669) who carried out the project work under my supervision.

SIGNATURE

Dr.G.Abirami

CD Faculty

CSE

SRM Institute of Science and Technology,

Potheri, SRM Nagar, Kattankulathur

Tamil Nadu 603203

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy**, for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal**, for bringing out novelty in all executions.

We would like to express my heartfelt thanks to **Chairperson, School of Computing Dr. Revathi Venkataraman**, for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professor** for their constant encouragement and support.

We are highly thankful to our Course project Internal guide **Dr.G.Abirami , Compiler Design Faculty , CSE**, for her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to the **Dr. M. PUSHPALATHA, Ph.D HEAD OF THE DEPARTMENT** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project

TABLE OF CONTENT

COMMAND LINE CALCULATOR

18CSC304J – Compiler Design Mini Project Report

TEAM MEMBERS:

- GELLE THARUN – RA2011003010660
- ANGADA CHANDRA MOULI – RA2011003010664
- EKKULURI RAJESH – RA2011003010669

INTRODUCTION

A command line calculator which supports mathematical expressions with scientific functions is very useful for most developers. The calculator available with Windows does not support most scientific functions The most difficult part we found when designing such a calculator was the parsing logic. Later while working with .NET, the runtime source code compilation made the parsing logic easy and interesting. It uses runtime compilation and saves the variables by serializing in a file. Thus, you can get the values of all the variables used in the previous Calculation.

PROBLEM STATEMENT

The calculate function calculates an expression. It uses the saved variables. I

have generated code which has a declaration of the variables

- To Evaluate the given expressions.
- To perform basic calculations

OBJECTIVES

The command line calculator is to be capable of parsing a human-readable mathematical expression with units, return the value if it can be evaluated and inform the user about the position of an error if not.

REQUIREMENTS

SOFTWARE REQUIREMENTS:

- Windows/Ubuntu Operating System
- C Programming Language

HARDWARE REQUIREMENTS:

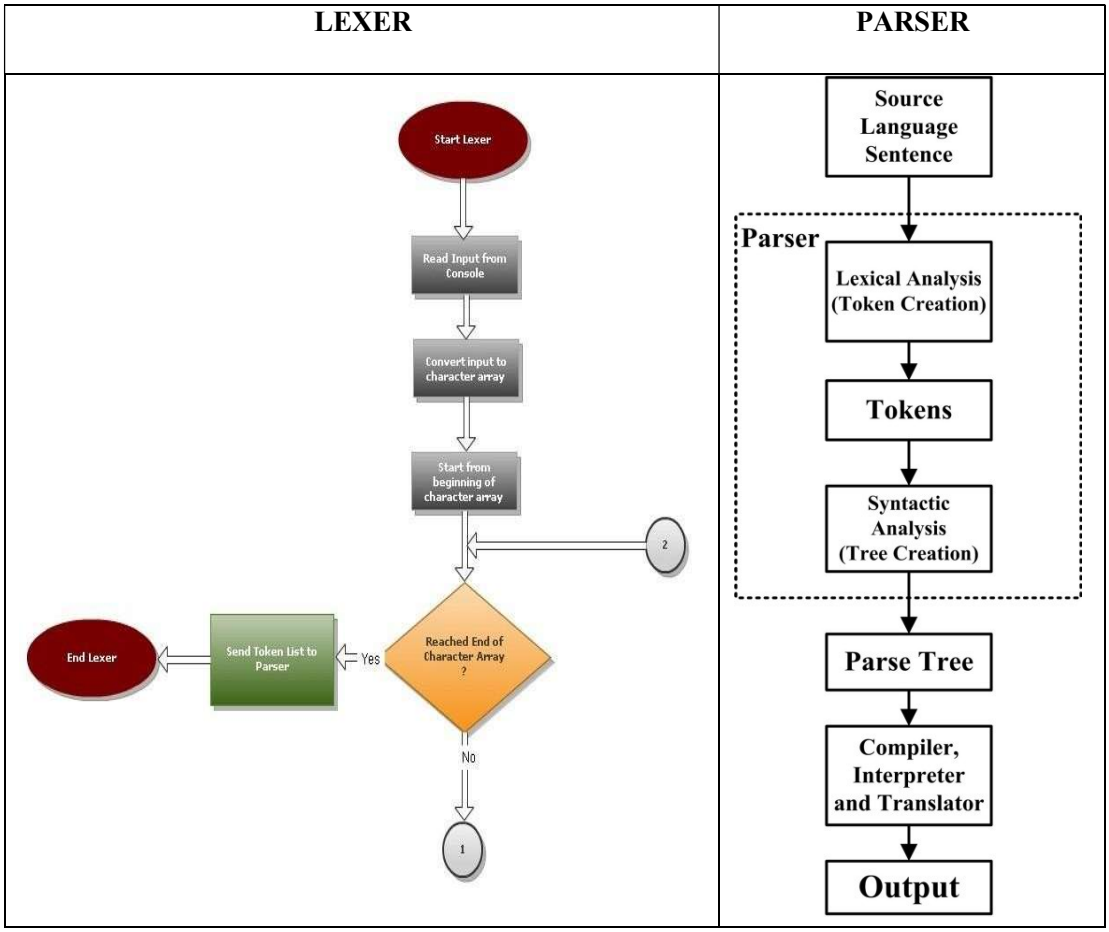
- Minimum of 4GB RAM

LEXER

- A LEXER is a software program that performs lexical analysis.
- Lexical analysis is the process of separating a stream of characters into different words, which in computer science we call 'tokens' .
- For Example -While reading we are performing the lexical operation of breaking the string of text at the space characters into multiple words.

PARSER

- A parser goes one level further than the lexer and takes the tokens produced by the lexer and tries to determine if proper sentences have been formed.
- Parsers work at the grammatical level, lexers work at the word level.
- Generally Yacc is used to parse language syntax.
- Yacc uses a parsing technique known as LR-parsing or shift-reduce parsing.



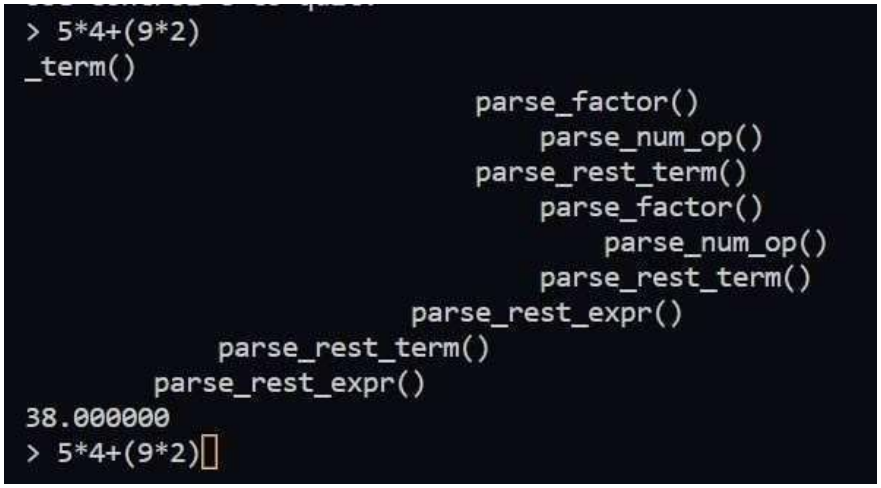
IMPLEMENTATION AND SOURCE CODE

- Implemented in C
- Source Code: https://github.com/neil-03/CLI-calculator

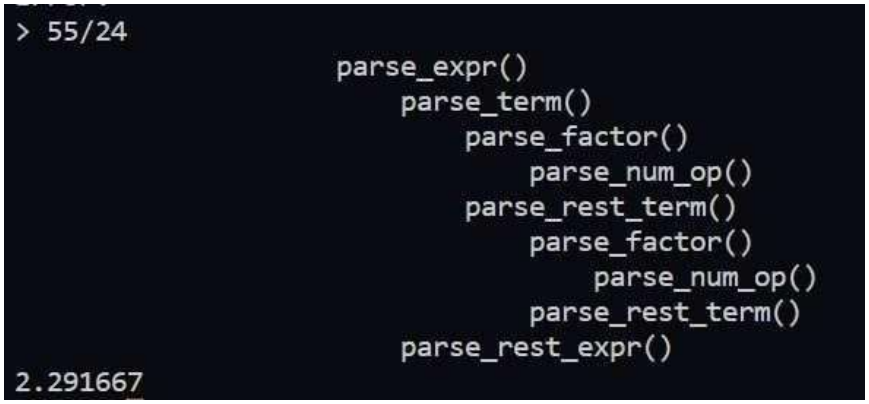
ALGORITHM

```
/*
expr := var rest_var
        term rest_expr
rest_expr := + term rest_expr
             - term rest_expr
             (nil)
term := factor rest_term
rest_term := * factor rest_term
             / factor rest_term
             % factor rest_term
             <nil>
factor := - factor
          num_op
num_op := num rest_num_op
          variable rest_num_op
          ( expr )  rest_num_op
rest_num_op := ^ num_op rest_num_op
               <nil>
rest_var := '=' expr
            rest_num_op
*/
```

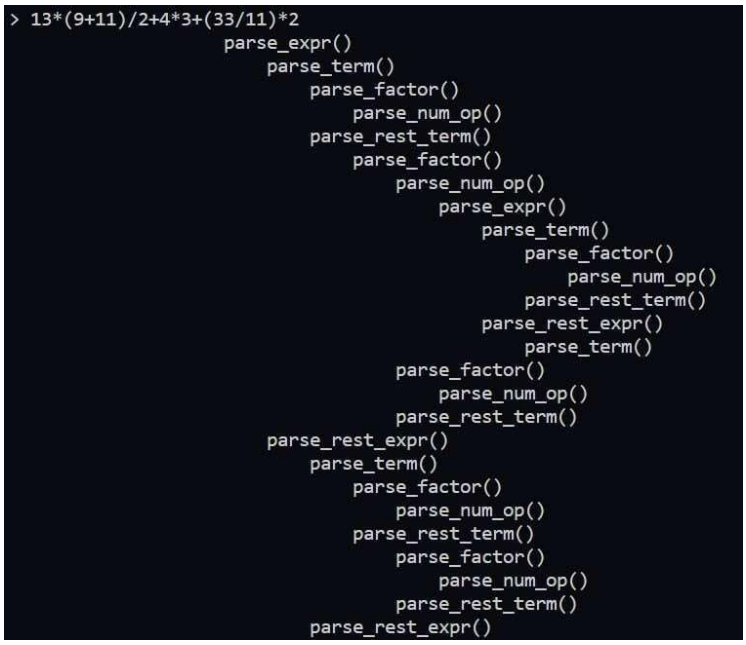
DEMO AND RESULTS

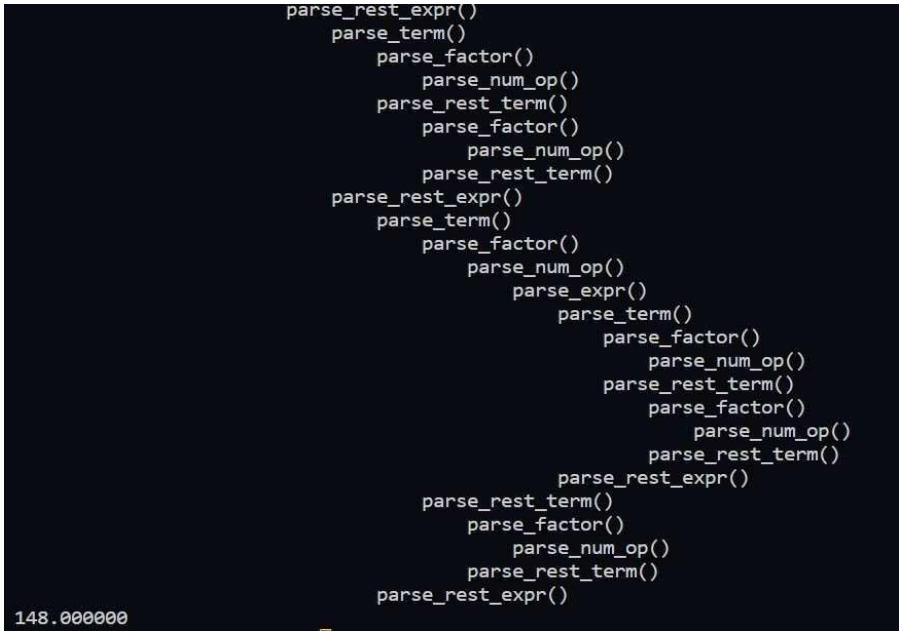


5*4+(9*2) = 38.000000



55/24 = 2.291667





13*(9+11)/2+4*3+(33/11)*2 = 148.000000

CONCLUSION

This is a powerful and versatile command-line calculator that really lives up to your expectation. Preloaded on all modern Linux distributions, this can make your number crunching tasks much easier to handle without leaving your terminals. Besides, if your shell script requires floating point calculation, can easily be invoked by the script to get the job done. All in all, CLC should definitely be in your productivity tool set.

REFERENCES

[1] - https://www.codeproject.com/Articles/12395/A-Command-Line-Calculator

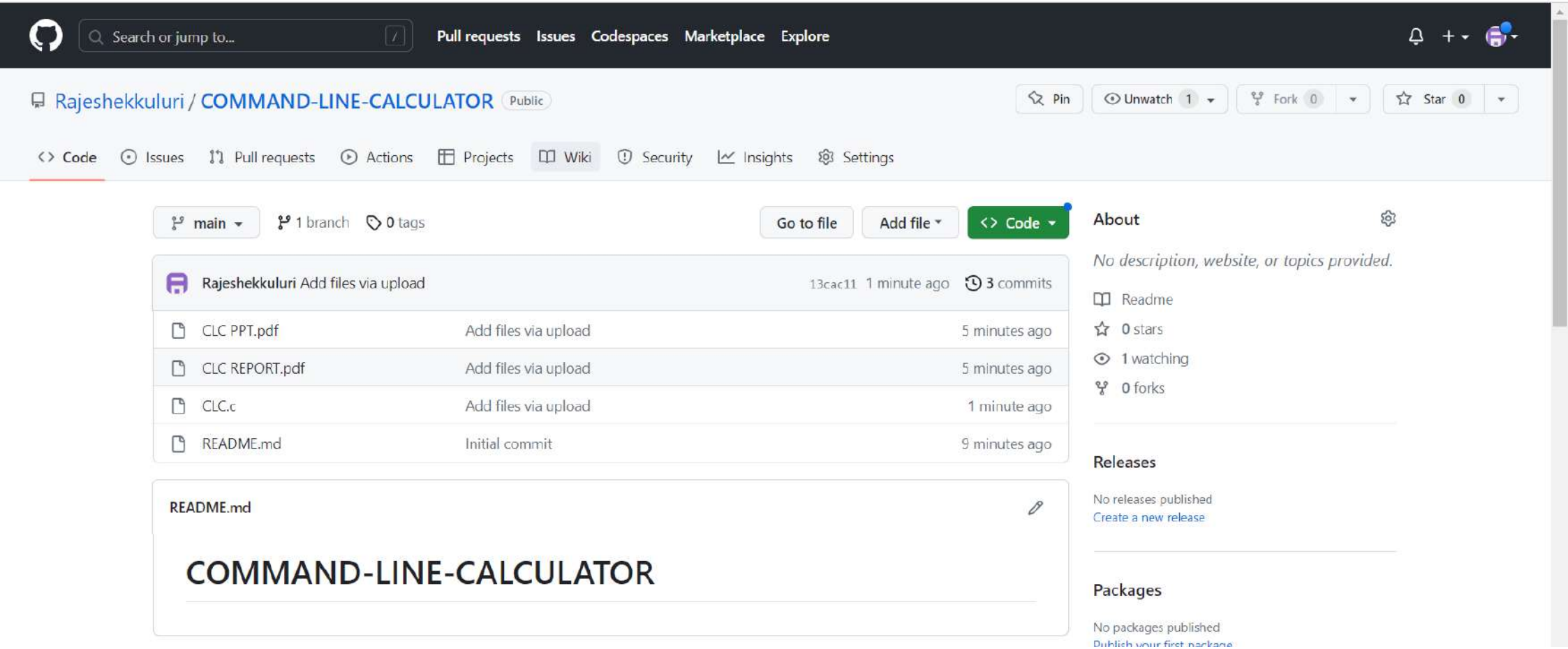
[2] - https://fedoramagazine.org/bc-command-line-calculator/

SUBMITTED BY

GELLE THARUN – RA2011003010660

ANGADA CHANDRA MOULI – RA2011003010664

EKKULURI RAJESH – RA2011003010669


Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
Rajeshekkuluri / COMMAND-LINE-CALCULATOR
Public
Pin
Unwatch 1
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
1 branch
0 tags
Go to file
Add file
Code
Rajeshekkuluri Add files via upload
13cac11 1 minute ago
3 commits
CLC PPT.pdf Add files via upload 5 minutes ago
CLC REPORT.pdf Add files via upload 5 minutes ago
CLC.c Add files via upload 1 minute ago
README.md Initial commit 9 minutes ago
README.md
COMMAND-LINE-CALCULATOR
About
No description, website, or topics provided.
Readme
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published

COMMAND LINE CALCULATOR

18CSC304J – Compiler Design

TEAM MEMBERS

- GELLE THARUN – RA2011003010660
- ANGADA CHANDRA MOULI – RA2011003010664
- EKKULURI RAJESH – RA2011003010669

Introduction

- A command line calculator which supports mathematical expressions with scientific functions is very useful for most developers.
- The calculator available with Windows does not support most scientific functions
- The most difficult part we found when designing such a calculator was the parsing logic.
- Later while working with .NET, the runtime source code compilation made the parsing logic easy and interesting.
- It uses runtime compilation and saves the variables by serializing in a file.
- Thus you can get the values of all the variables used in the previous Calculation.

Problem Statement

The calculate function calculates an expression. It uses the saved variables. I have generated code which has a declaration of the variables

- To Evaluate the given expressions.
- To perform basic calculations

Objectives

The command line calculator is to be capable of parsing a human-readable mathematical expression with units, return the value if it can be evaluated and inform the user about the position of an error if not.

Requirements

SOFTWARE REQUIREMENTS:

- Windows/Ubuntu Operating System
- C Programming Language

HARDWARE REQUIREMENTS :

- Minimum of 4GB RAM

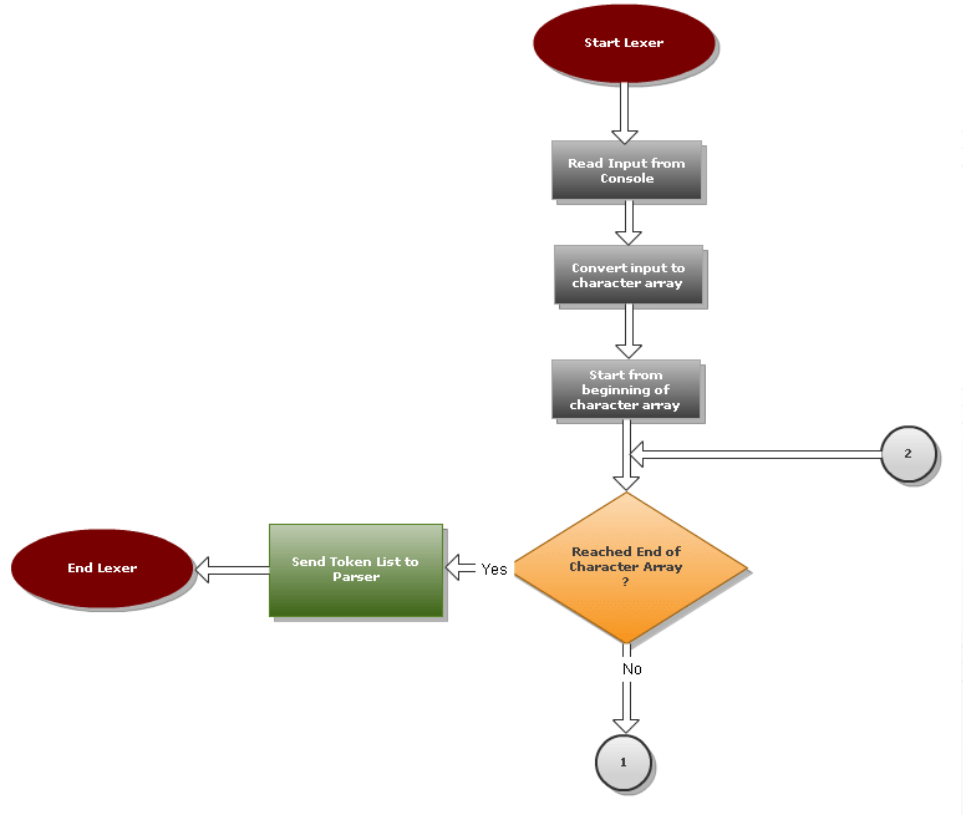
LEXER

- A LEXER is a software program that performs lexical analysis.
- Lexical analysis is the process of separating a stream of characters into different words, which in computer science we call 'tokens' .
- For Example -While reading we are performing the lexical operation of breaking the string of text at the space characters into multiple words.

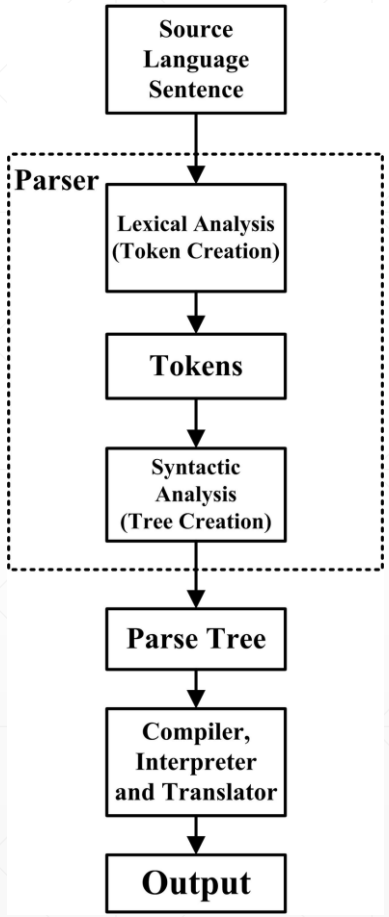
PARSER

- A parser goes one level further than the lexer and takes the tokens produced by the lexer and tries to determine if proper sentences have been formed.
- Parsers work at the grammatical level, lexers work at the word level.
- Generally yacc is used to parse language syntax.
- Yacc uses a parsing technique known as LR-parsing or shift-reduce parsing.

LEXER



PARSER

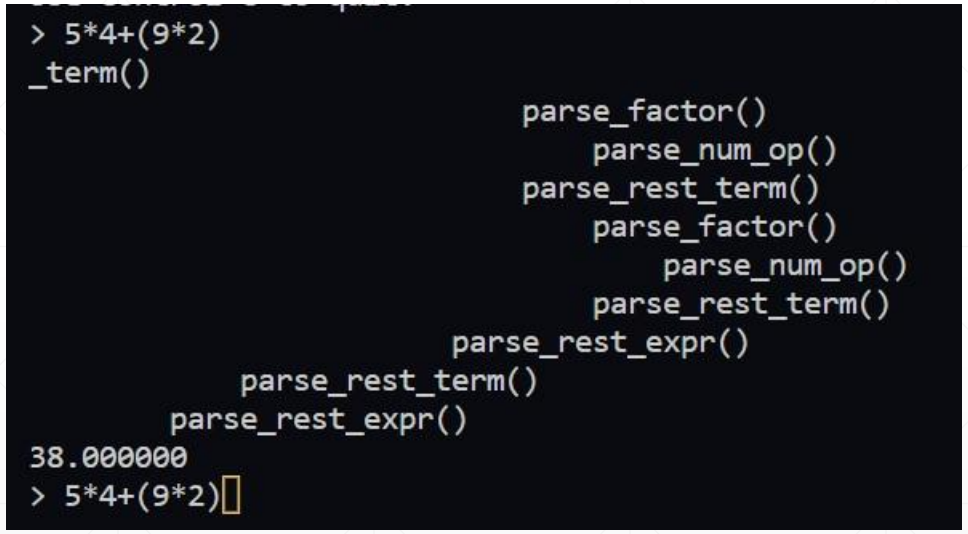


Implementation and Source Code

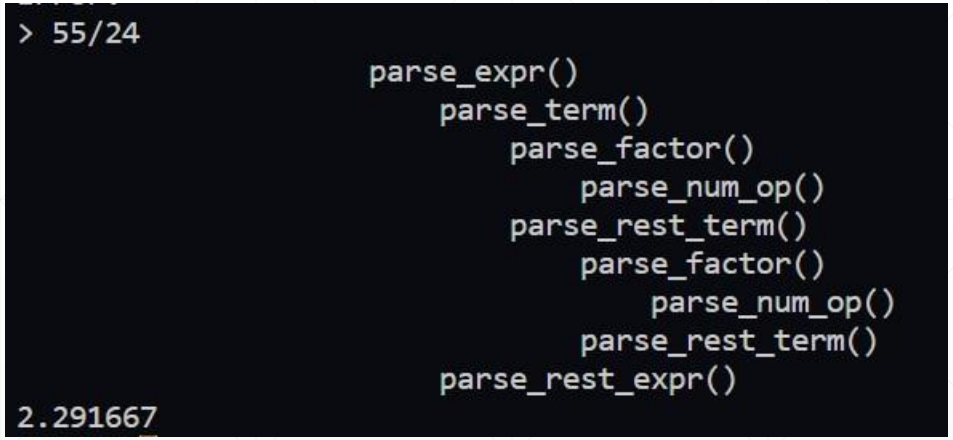
- Implemented in C
- Source Code: https://github.com/neil-03/CLI-calculator

Demo and Results

5*4+(9+2) = 38.000000

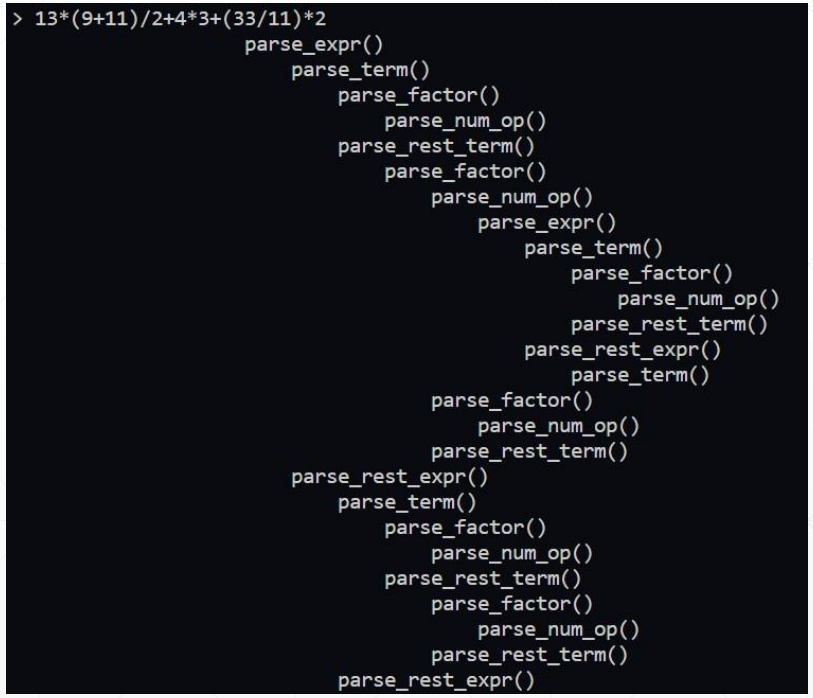


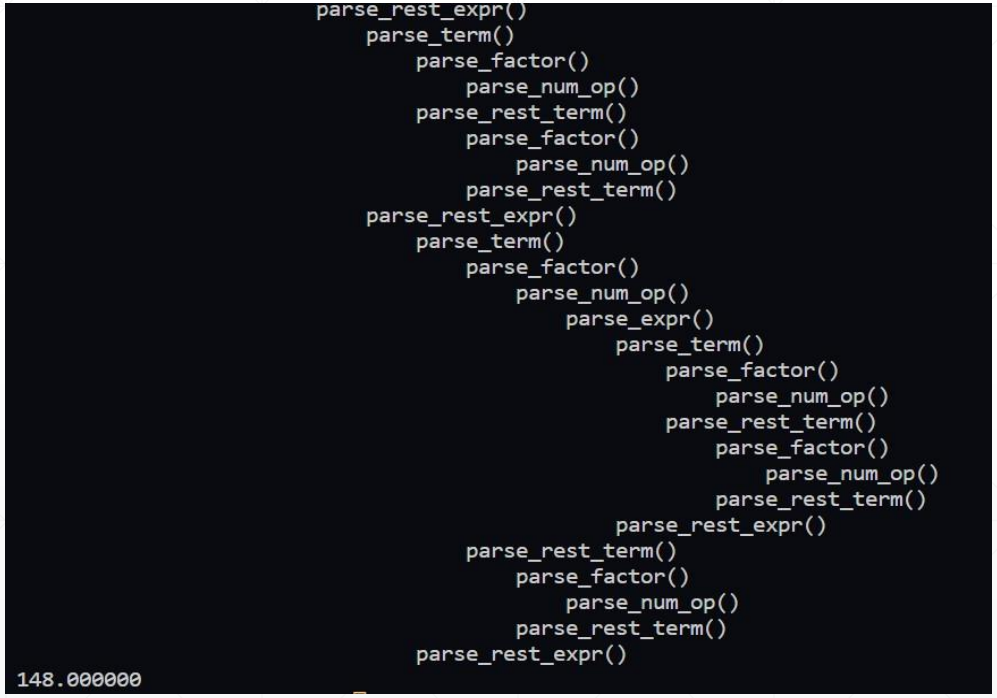
55/24 = 2.291667



Demo and Results (contd…)

13*(9+11)/2+4*3+(33/11)*2 = 148.000000





Conclusion

This is a powerful and versatile command-line calculator that really lives up to your expectation. Preloaded on all modern Linux distributions, this can make your number crunching tasks much easier to handle without leaving your terminals. Besides, if your shell script requires floating point calculation, can easily be invoked by the script to get the job done. All in all, CLC should definitely be in your productivity tool set.

References

[1] - https://www.codeproject.com/Articles/12395/A-Command-Line-Calculator

[2] - https://fedoramagazine.org/bc-command-line-calculator/

THANK YOU

GELLE THARUN – RA2011003010660

ANGADA CHANDRA MOULI-RA2011003010664

EKKULURI RAJESH – RA2011003010669

LEXICAL ANALYZER

EX. NO. 1

AIM: To write a program to implement a lexical analyzer.

ALGORITHM:

1. Start.
2. Get the input program from the file prog.txt.
3. Read the program line by line and check if each word in a line is a keyword, identifier, constant or an operator.
4. If the word read is an identifier, assign a number to the identifier and make an entry into the symbol table stored in sybol.txt.
5. For each lexeme read, generate a token as follows:

a. If the lexeme is an identifier, then the token generated is of the form <id, number>

b. If the lexeme is an operator, then the token generated is <op, operator>.

c. If the lexeme is a constant, then the token generated is <const, value>.

d. If the lexeme is a keyword, then the token is the keyword itself.

6. The stream of tokens generated are displayed in the console output.
7. Stop.

PROGRAM:

```
file = open("add.c", 'r')
lines = file.readlines()

keywords    = ["void", "main", "int", "float", "bool", "if", "for", "else", "while", "char", "return"]
operators = ["=", "==", "+", "-", "*", "/", "++", "--", "+=", "-=", "!=", "||", "&&"]
punctuations= [";", "(", ")", "{", "}", "[", "]"]

def is_int(x):
    try:
```

```
        int(x)
        return True
    except:
        return False

for line in lines:
    for i in line.strip().split(" "):
        if i in keywords:
            print (i, " is a keyword")
        elif i in operators:
            print (i, " is an operator")
        elif i in punctuations:
            print (i, " is a punctuation")
        elif is_int(i):
            print (i, " is a number")
        else:
            print (i, " is an identifier")
```

INPUT :

```
#include <stdio.h>

void main ( )

{
   int x = 6 ;
   int y = 4 ;
   x = x + y ;
}
```

OUTPUT :

```
#include  is an identifier
<stdio.h>  is an identifier
  is an identifier
void  is a keyword
main  is a keyword
(  is a punctuation
)  is a punctuation
  is an identifier
{  is a punctuation
int  is a keyword
x  is an identifier
=  is an operator
6  is a number
;  is a punctuation
int  is a keyword
y  is an identifier
=  is an operator
4  is a number
;  is a punctuation
x  is an identifier
=  is an operator
x  is an identifier
+  is an operator
y  is an identifier
;  is a punctuation
}  is a punctuation
```

RESULT :

The implementation of lexical analyser in C++ was compiled, executed and verified successfully.

CONVERSION FROM REGULAR EXPRESSION TO NFA

EX. NO. 2

AIM: To write a program for converting Regular Expression to NFA.

ALGORITHM:

1. Start
2. Get the input from the user
3. Initialize separate variables and functions for Postfix , Display and NFA
4. Create separate methods for different operators like +,*, .
5. By using Switch case Initialize different cases for the input
6. For ' . ' operator Initialize a separate method by using various stack functions do the same for the other operators like ' * ' and ' + '.
7. Regular expression is in the form like a.b (or) a+b
8. Display the output
9. Stop

PROGRAM :

```
transition_table = [ [0]*3 for _ in range(20) ]

re = input("Enter the regular expression : ")
re += " "

i = 0
j = 1
while(i<len(re)):
    if re[i] == 'a':
        try:
            if re[i+1] != '|' and re[i+1] !='*':
```

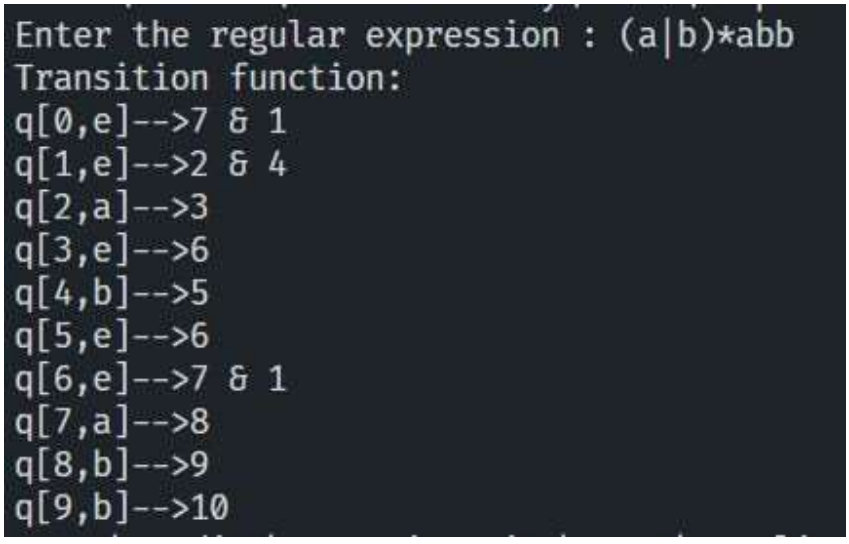
```
            transition_table[j][0] = j+1
            j += 1
        elif re[i+1] == '|' and re[i+2] =='b':
            transition_table[j][2]=((j+1)*10)+(j+3)
            j+=1
            transition_table[j][0]=j+1
            j+=1
            transition_table[j][2]=j+3
            j+=1
            transition_table[j][1]=j+1
            j+=1
            transition_table[j][2]=j+1
            j+=1
            i=i+2
        elif re[i+1]=='*':
            transition_table[j][2]=((j+1)*10)+(j+3)
            j+=1
            transition_table[j][0]=j+1
            j+=1
            transition_table[j][2]=((j+1)*10)+(j-1)
            j+=1
    except:
        transition_table[j][0] = j+1

elif re[i] == 'b':
    try:
        if re[i+1] != '|' and re[i+1] !='*':
            transition_table[j][1] = j+1
            j += 1
        elif re[i+1]=='|' and re[i+2]=='a':
            transition_table[j][2]=((j+1)*10)+(j+3)
```

```
            j+=1
            transition_table[j][1]=j+1
            j+=1
            transition_table[j][2]=j+3
            j+=1
            transition_table[j][0]=j+1
            j+=1
            transition_table[j][2]=j+1
            j+=1
            i=i+2
        elif re[i+1]=='*':
            transition_table[j][2]=((j+1)*10)+(j+3)
            j+=1
            transition_table[j][1]=j+1
            j+=1
            transition_table[j][2]=((j+1)*10)+(j-1)
            j+=1
    except:
        transition_table[j][1] = j+1

elif re[i]=='e' and re[i+1]!='|'and re[i+1]!='*':
    transition_table[j][2]=j+1
    j+=1

elif re[i]==')' and re[i+1]=='*':

    transition_table[0][2]=((j+1)*10)+1
    transition_table[j][2]=((j+1)*10)+1
    j+=1

i +=1
```

```
print ("Transition function:")
for i in range(j):
    if(transition_table[i][0]!=0):
        print("q[{0},a]-->{1}".format(i,transition_table[i][0]))
    if(transition_table[i][1]!=0):
        print("q[{0},b]-->{1}".format(i,transition_table[i][1]))
    if(transition_table[i][2]!=0):
        if(transition_table[i][2]<10):
            print("q[{0},e]-->{1}".format(i,transition_table[i][2]))
        else:
            print("q[{0},e]-->{1} &
{2}".format(i,int(transition_table[i][2]/10),transition_table[i][2]%10))
```

INPUT : (a|b)*abb

OUTPUT :



RESULT :

The program to convert regular expressions to NFA was implemented successfully.

CONVERSION OF NFA TO DFA

EX. NO. 3

AIM: To write a program for converting NFA to DFA.

ALGORITHM:

1. Start
2. Get the input from the user
3. Set the only state in SDFA to “unmarked”.
4. while SDFA contains an unmarked state do:

a. Let T be that unmarked state

b. for each a in % do S = e-Closure(MoveNFA(T,a))

c. if S is not in SDFA already then, add S to SDFA (as an “unmarked” state)

d. Set MoveDFA(T,a) to S

5. For each S in SDFA if any s & S is a final state in the NFA then, mark S an a final state in the DFA
6. Print the result.
7. Stop the program

PROGRAM:

```
import pandas as pd


nfa = {}
n = int(input("No. of states : "))
t = int(input("No. of transitions : "))
for i in range(n):
    state = input("state name : ")
    nfa[state] = {}
    for j in range(t):
        path = input("path : ")
```

```
        print("Enter end state from state {} travelling through path {} : ".format(state, path))
        reaching_state = [x for x in input().split()]
        nfa[state][path] = reaching_state

print("\nNFA :- \n")
print(nfa)
print("\nPrinting NFA table :- ")
nfa_table = pd.DataFrame(nfa)
print(nfa_table.transpose())

print("Enter final state of NFA : ")
nfa_final_state = [x for x in input().split()]

new_states_list = []
dfa = {}
keys_list = list(
    list(nfa.keys())[0])
path_list = list(nfa[keys_list[0]].keys())

dfa[keys_list[0]] = {}
for y in range(t):
    var = "".join(nfa[keys_list[0]][
                    path_list[y]])
    dfa[keys_list[0]][path_list[y]] = var
    if var not in keys_list:
        new_states_list.append(var)
        keys_list.append(var)

while len(new_states_list) != 0:
    dfa[new_states_list[0]] = {}
    for _ in range(len(new_states_list[0])):
```

```
        for i in range(len(path_list)):
            temp = []
            for j in range(len(new_states_list[0])):
                temp += nfa[new_states_list[0][j]][path_list[i]]
            s = ""
            s = s.join(temp)
            if s not in keys_list:
                new_states_list.append(s)
                keys_list.append(s)
            dfa[new_states_list[0]][path_list[i]] = s

    new_states_list.remove(new_states_list[0])

print("\nDFA :- \n")
print(dfa)
print("\nPrinting DFA table :- ")
dfa_table = pd.DataFrame(dfa)
print(dfa_table.transpose())

dfa_states_list = list(dfa.keys())
dfa_final_states = []
for x in dfa_states_list:
    for i in x:
        if i in nfa_final_state:
            dfa_final_states.append(x)
            break

print("\nFinal states of the DFA are : ", dfa_final_states)
```

INPUT :

No. of states : 3

No. of transitions : 2

state name : A

path : 0
Enter end state from state A travelling through path 0 :

A

path : 1
Enter end state from state A travelling through path 1 :

A B

state name : B

path : 0
Enter end state from state B travelling through path 0 :

C

path : 1
Enter end state from state B travelling through path 1 :

C

state name : C

path : 0

Enter end state from state C travelling through path 0 :

path : 1

Enter end state from state C travelling through path 1 :

NFA :-

{'A': {'0': ['A'], '1': ['A', 'B']}, 'B': {'0': ['C'], '1': ['C']}, 'C': {'0': [], '1': []}}

Printing NFA table :-

	0	1
A	[A]	[A, B]
B	[C]	[C]
C	[]	[]

Enter final state of NFA :

C

OUTPUT :

```
DFA :-

{'A': {'0': 'A', '1': 'AB'}, 'AB': {'0': 'AC', '1': 'ABC'}, 'AC': {'0': 'A', '1': 'AB'}, 'ABC': {'0': 'AC', '1': 'ABC'}}

Printing DFA table :-
      0    1
A     A   AB
AB   AC  ABC
AC    A   AB
ABC  AC  ABC

Final states of the DFA are :  ['AC', 'ABC']
```

RESULT :

The given NFA was converted to a DFA using python successfully.

ELIMINATION OF LEFT RECURSION

EX. NO. 4(a)

AIM: A program for Elimination of Left Recursion.

ALGORITHM:

1. Start the program.
2. Initialize the arrays for taking input from the user.
3. Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.
4. Prompt the user to input the production for non-terminals.
5. Eliminate left recursion using the following rules:-

   A->Aα1| Aα2 | .....|Aαm

   A->β1| β2| .....| βn

   Then replace it by

   A-> βi A' i=1,2,3,… .m

   A'-> αj A' j=1,2,3,….n

   A'-> Ɛ
6. After eliminating the left recursion by applying these rules, display the productions without left recursion.
7. Stop.

PROGRAM:

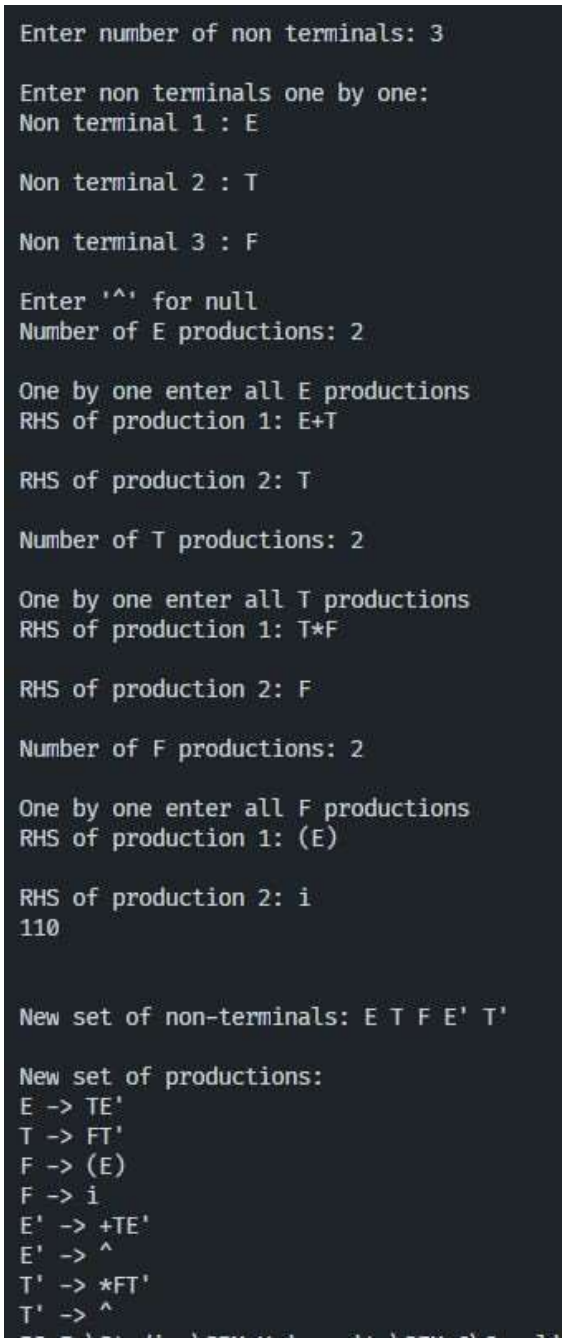
```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main()
{
```

```
int n;
cout<<"\nEnter number of non terminals: ";
cin>>n;
cout<<"\nEnter non terminals one by one: ";
int i;
vector<string> nonter(n);
vector<int>  leftrecr(n,0);
for(i=0;i<n;++i) {
      cout<<"\nNon terminal "<<i+1<<" : ";
   cin>>nonter[i];
}
vector<vector<string> > prod;
cout<<"\nEnter '^' for null";
for(i=0;i<n;++i) {
   cout<<"\nNumber of "<<nonter[i]<<" productions: ";
   int k;
   cin>>k;
   int j;
   cout<<"\nOne by one enter all "<<nonter[i]<<" productions";
   vector<string> temp(k);
   for(j=0;j<k;++j) {
      cout<<"\nRHS of production "<<j+1<<": ";
      string abc;
      cin>>abc;
      temp[j]=abc;

if(nonter[i].length()<=abc.length()&&nonter[i].compare(abc.substr(0,nonter[i].length()))==0)
         leftrecr[i]=1;
   }
   prod.push_back(temp);
}
```

```
    for(i=0;i<n;++i) {
        cout<<leftrecr[i];
    }
    for(i=0;i<n;++i) {
        if(leftrecr[i]==0)
            continue;
        int j;
        nonter.push_back(nonter[i]+"'");
        vector<string> temp;
        for(j=0;j<prod[i].size();++j) {

if(nonter[i].length()<=prod[i][j].length()&&nonter[i].compare(prod[i][j].substr(0,nonter[i].length
()))==0) {
                string
abc=prod[i][j].substr(nonter[i].length(),prod[i][j].length()-nonter[i].length())+nonter[i]+"'";
                temp.push_back(abc);
                prod[i].erase(prod[i].begin()+j);
                --j;
            }
            else {
                prod[i][j]+=nonter[i]+"'";
            }
        }
        temp.push_back("^");
        prod.push_back(temp);
    }
    cout<<"\n\n";
    cout<<"\nNew set of non-terminals: ";
    for(i=0;i<nonter.size();++i)
        cout<<nonter[i]<<" ";
    cout<<"\n\nNew set of productions: ";
```

```
    for(i=0;i<nonter.size();++i)  {
        int j;
        for(j=0;j<prod[i].size();++j) {
            cout<<"\n"<<nonter[i]<<" -> "<<prod[i][j];
        }
    }
    return 0;
}
```

OUTPUT :



```
Enter number of non terminals: 3

Enter non terminals one by one:
Non terminal 1 : E

Non terminal 2 : T

Non terminal 3 : F

Enter '^' for null
Number of E productions: 2

One by one enter all E productions
RHS of production 1: E+T

RHS of production 2: T

Number of T productions: 2

One by one enter all T productions
RHS of production 1: T*F

RHS of production 2: F

Number of F productions: 2

One by one enter all F productions
RHS of production 1: (E)

RHS of production 2: i
110


New set of non-terminals: E T F E' T'

New set of productions:
E -> TE'
T -> FT'
F -> (E)
F -> i
E' -> +TE'
E' -> ^
T' -> *FT'
T' -> ^
```

RESULT :

A program for Elimination of Left Recursion was run successfully.

LEFT FACTORING

EX. NO. 4(b)

AIM : A program for implementation Of Left Factoring

ALGORITHM :

1. Start
2. Ask the user to enter the set of productions
3. Check for common symbols in the given set of productions by comparing with:

   A->aB1|aB2
4. If found, replace the particular productions with:

   A->aA'

   A'->B1 | B2|ε
5. Display the output
6. Exit

CODE :

```
#include <iostream>
#include <math.h>
#include <vector>
#include <string>
#include <stdlib.h>
using namespace std;

int main()
{
   cout<<"\nEnter number of productions: ";
   int p;
   cin>>p;
   vector<string> prodleft(p),prodright(p);
```

```
cout<<"\nEnter productions one by one: ";
int i;
for(i=0;i<p;++i) {
   cout<<"\nLeft of production "<<i+1<<": ";
   cin>>prodleft[i];
   cout<<"\nRight of production "<<i+1<<": ";
   cin>>prodright[i];
}
int  j;
int e=1;
for(i=0;i<p;++i) {
   for(j=i+1;j<p;++j) {
      if(prodleft[j]==prodleft[i]) {
         int k=0;
         string com="";

while(k<prodright[i].length()&&k<prodright[j].length()&&prodright[i][k]==prodright[j][k]) {
            com+=prodright[i][k];
            ++k;
         }
         if(k==0)
            continue;
         char* buffer;
         string comleft=prodleft[i];
         if(k==prodright[i].length()) {
            prodleft[i]+=string(itoa(e,buffer,10));
            prodleft[j]+=string(itoa(e,buffer,10));
            prodright[i]="^";
            prodright[j]=prodright[j].substr(k,prodright[j].length()-k);
         }
         else if(k==prodright[j].length()) {
```

```
            prodleft[i]+=string(itoa(e,buffer,10));
            prodleft[j]+=string(itoa(e,buffer,10));
            prodright[j]="^";
            prodright[i]=prodright[i].substr(k,prodright[i].length()-k);
          }
          else {
            prodleft[i]+=string(itoa(e,buffer,10));
            prodleft[j]+=string(itoa(e,buffer,10));
            prodright[j]=prodright[j].substr(k,prodright[j].length()-k);
            prodright[i]=prodright[i].substr(k,prodright[i].length()-k);
          }
          int l;
          for(l=j+1;l<p;++l) {

if(comleft==prodleft[l]&&com==prodright[l].substr(0,fmin(k,prodright[l].length()))) {
              prodleft[l]+=string(itoa(e,buffer,10));
              prodright[l]=prodright[l].substr(k,prodright[l].length()-k);
            }
          }
          prodleft.push_back(comleft);
          prodright.push_back(com+prodleft[i]);
          ++p;
          ++e;
        }
      }
    }
    cout<<"\n\nNew productions";
    for(i=0;i<p;++i) {
      cout<<"\n"<<prodleft[i]<<"->"<<prodright[i];
    }
    return 0; }
```

OUTPUT :

```
Enter the no. of nonterminals : 2

Nonterminal 1
Enter the no. of productions : 3

Enter LHS : S
S->iCtSeS
S->iCtS
S->a

Nonterminal 2
Enter the no. of productions : 1

Enter LHS : C
C->b


The resulting productions are :

 S' -> ε| eS |  |

 C ->  b

 S ->  iCtSS' | a
```

RESULT : A program for implementation Of Left Factoring was compiled and run successfully.

FIRST AND FOLLOW

AIM: To write a program to perform first and follow using any language.

ALGORITHM:

For computing the first:

1. If X is a terminal then FIRST(X) = {X}

Example: F -> I | id

We can write it as FIRST(F) -> { (, id)

2. If X is a non-terminal like E -> T then to get FIRSTI substitute T with other productions until you get a terminal as the first symbol

3. If X -> ε then add ε to FIRST(X).

For computing the follow:

1. Always check the right side of the productions for a non-terminal, whose FOLLOW set is being found. (never see the left side).

2. (a) If that non-terminal (S,A,B…) is followed by any terminal (a,b…,*,+,(,)…) , then add that terminal into the FOLLOW set.

(b) If that non-terminal is followed by any other non-terminal then add FIRST of other nonterminal into the FOLLOW set.

CODE :

```
import re
import string
import pandas as pd


def parse(user_input,start_symbol,parsingTable):

    #flag
    flag = 0

    #appending dollar to end of input
    user_input = user_input + "$"

    stack = []

    stack.append("$")
    stack.append(start_symbol)

    input_len = len(user_input)
    index = 0


    while len(stack) > 0:

        #element at top of stack
        top = stack[len(stack)-1]

        print ("Top =>",top)

        #current input
```

```
current_input = user_input[index]

print ("Current_Input => ",current_input)

if top == current_input:
        stack.pop()
        index = index + 1
else:

        #finding value for key in table
        key = top , current_input
        print (key)

        #top of stack terminal => not accepted
        if key not in parsingTable:
                flag = 1
                break

        value = parsingTable[key]
        if value !='@':
                value = value[::-1]
                value = list(value)

                #poping top of stack
                stack.pop()

                #push value chars to stack
                for element in value:
                        stack.append(element)
        else:
                stack.pop()
```

```
    if flag == 0:
        print ("String accepted!")
    else:
        print ("String not accepted!")



def ll1(follow, productions):

    print ("\nParsing Table\n")

    table = {}
    for key in productions:
        for value in productions[key]:
            if value!='@':
                for element in first(value, productions):
                    table[key, element] = value
            else:
                for element in follow[key]:
                    table[key, element] = value

    for key,val in table.items():
        print (key,"=>",val)

    new_table = {}
    for pair in table:
        new_table[pair[1]] = {}

    for pair in table:
        new_table[pair[1]][pair[0]] = table[pair]
```

```
        print ("\n")
        print ("\nParsing Table in matrix form\n")
        print (pd.DataFrame(new_table).fillna('-'))
        print ("\n")

        return table

def follow(s, productions, ans):
        if len(s)!=1 :
                return {}

        for key in productions:
                for value in productions[key]:
                        f = value.find(s)
                        if f!=-1:
                                if f==(len(value)-1):
                                        if key!=s:
                                                if key in ans:
                                                        temp = ans[key]
                                                else:
                                                        ans = follow(key, productions, ans)
                                                        temp = ans[key]
                                                ans[s] = ans[s].union(temp)
                                else:
                                        first_of_next = first(value[f+1:], productions)
                                        if '@' in first_of_next:
                                                if key!=s:
                                                        if key in ans:
                                                                temp = ans[key]
```

```
                                        else:
                                            ans = follow(key, productions, ans)
                                            temp = ans[key]
                                        ans[s] = ans[s].union(temp)
                                        ans[s] = ans[s].union(first_of_next) - {'@'}
                                else:
                                    ans[s] = ans[s].union(first_of_next)
        return ans

def first(s, productions):
        c = s[0]
        ans = set()
        if c.isupper():
                for st in productions[c]:
                        if st == '@' :
                                if len(s)!=1 :
                                        ans = ans.union( first(s[1:], productions) )
                                else :
                                        ans = ans.union('@')
                        else :
                                f = first(st, productions)
                                ans = ans.union(x for x in f)
        else:
                ans = ans.union(c)
        return ans

if __name__=="__main__":
        productions=dict()
        grammar = open("grammar2", "r")
        first_dict = dict()
        follow_dict = dict()
```

```
flag = 1
start = ""
for line in grammar:
        l = re.split("( |->|\n|\|)*", line)
        lhs = l[0]
        rhs = set(l[1:-1])-{''}
        if flag :
                flag = 0
                start = lhs
        productions[lhs] = rhs

print ('\nFirst\n')
for lhs in productions:
        first_dict[lhs] = first(lhs, productions)
for f in first_dict:
        print (str(f) + " : " + str(first_dict[f]))
print ("")

print ('\nFollow\n')

for lhs in productions:
        follow_dict[lhs] = set()

follow_dict[start] = follow_dict[start].union('$')

for lhs in productions:
        follow_dict = follow(lhs, productions, follow_dict)

for lhs in productions:
        follow_dict = follow(lhs, productions, follow_dict)
```

```
for f in follow_dict:
        print (str(f) + " : " + str(follow_dict[f]))

ll1Table = ll1(follow_dict, productions)

#parse("a*(a+a)",start,ll1Table)
parse("ba=a+23",start,ll1Table)

# tp(ll1Table)
```

OUTPUT :

```
Enter the productions(type 'end' at the last of the production)
E->TA
A->+TA
A->ε
T->FB
B->*FB
B->ε
F->(E)
F->i
end
E       (i      $)
A       +ε      $)
T       (i      +$)
B       *ε      +$)
F       (i      *+$)
```

RESULT: The FIRST and FOLLOW sets of the non-terminals of a grammar were found successfully using python language.

PREDICTIVE PARSING

Aim: A program for Predictive Parsing

Algorithm:-

1. Start the program.
2. Initialize the required variables.
3. Get the number of coordinates and productions from the user.
4. Perform the following

for (each production A → α in G) {for
(each terminal a in FIRST(α))add A
→ α to M[A, a];
if (ε is in FIRST(α))
for (each symbol b in FOLLOW(A))
add A → α to M[A, b];

5. Print the resulting stack.
6. Print if the grammar is accepted or not.
7. Exit the program.

Program:

```
#include<stdio.h
>
#include<conio.
h>
#include<string.
h>void main()
{
char fin[10][20],st[10][20],ft[20][20],fol[20][20];
int
a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;
clrscr();
printf("enter the no. of
nonterminals\n");scanf("%d",&n);
printf("enter the productions in a
grammar\n");for(i=0;i<n;i++)
scanf("%s",st[i
]);
for(i=0;i<n;i++
) fol[i][0]='\0';
for(s=0;s<n;s+
+)
{
for(i=0;i<n;i++)
{
j=3;
l=0;
a=0
;
l1:if(!((st[i][j]>64)&&(st[i][j]<91)))
{
for(m=0;m<l;m++)
{
if(ft[i][m]==st[i]
[j])goto s1;
}
ft[i][l]=st[i][
j];l=l+1;
```

```
s1:j=j+1;
}
else

{
if(s>0)
{
while(st[i][j]!=st[a][0])
{
a++;
}
b=0;
while(ft[a][b]!='\0')
{
for(m=0;m<l;m++)
{
if(ft[i][m]==ft[a][
b])goto s2;
}
ft[i][l]=ft[a][
b];l=l+1;
s2:b=b+1;
}
}
}
while(st[i][j]!='\0')
{
if(st[i][j]=='|')
{
j=j+1;
goto l1;
}
j=j+1;
}
ft[i][l]='\0';
}
}
printf("first
\n");
for(i=0;i<n;i++
)
printf("FIRS[%c]=%s\n",st[i][0],
ft[i]);fol[0][0]='$';
for(i=0;i<n;i++)
{
k=0
;
j=3;
if(i==
0)l=1;
else
l=0
;
k1:while((st[i][0]!=st[k][j])&&(k<n))
```

```
 {
 if(st[k][j]=='\0')
 {
 k++
 ;
 j=2;

}
j++;
 }
 j=j+1;
 if(st[i][0]==st[k][j-1])
 {
 if((st[k][j]!='|')&&(st[k][j]!='\0'))
 {
 a=0;
 if(!((st[k][j]>64)&&(st[k][j]<91)))
 {
 for(m=0;m<l;m++)
 {
 if(fol[i][m]==st[k]
 [j])goto q3;
 }
 fol[i][l]=st[k][
 j];l++;
 q3:
 }
 else
 {
 while(st[k][j]!=st[a][0])
 {
 a++;
 }
 p=0;
 while(ft[a][p]!='\0')
 {
 if(ft[a][p]!='@')
 {
 for(m=0;m<l;m++)
 {
 if(fol[i][m]==ft[a][
 p])goto q2;
 }
 fol[i][l]=ft[a][
 p];l=l+1;
 }
 else
 e=1
 ;
 q2:p++;
 }
 if(e==1)
 {
 e=0
```

```
;
goto a1;
}
}
}
else

{
a1:c=0
;a=0;
while(st[k][0]!=st[a][0])
{
a++;
}
while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))
{
for(m=0;m<l;m++)
{
if(fol[i][m]==fol[a]
[c])goto q1;
}
fol[i][l]=fol[a][
c];l++;
q1:c++;
}
}
goto k1;
}
fol[i][l]='\0';
}
printf("follow
\n");
for(i=0;i<n;i++)
printf("FOLLOW[%c]=%s\n",st[i][0],
fol[i]);printf("\n");
s=0;
for(i=0;i<n;i++)
{
j=3
;
while(st[i][j]!='\0')
{
if((st[i][j-1]=='|')||(j==3))
{
for(p=0;p<=2;p++)
{
fin[s][p]=st[i][p];
}
t=j;
for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));
p++)
{
fin[s][p]=st[i][
j];j++;
```

```
}
fin[s][p]='\0';
if(st[i][k]=='@')
{
b=0
;
a=0
;
while(st[a][0]!=st[i][0])

{
a++;
}
while(fol[a][b]!='\0')
{
printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],f
in[s]);b++;
}
}
else if(!((st[i][t]>64)&&(st[i][t]<91)))
printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fi
n[s]);else
{
b=0
;
a=0
;
while(st[a][0]!=st[i][3])
{
a++;
}
while(ft[a][b]!='\0')
{
printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fi
n[s]);b++;
}
}
s++
;
}
if(st[i][j]=='|
')j++;
}
}
getch();
}
```

Output:

```
Enter the no. of
nonterminals2
Enter the productions in a
grammarS->CC
C->eC |
dFirst
FIRS[S] =
edFIRS[C]
= ed
Follow
FOLLOW[S] =$
FOLLOW[C]
=ed$M [S , e]
=S->CC
M [S , d] =S->CC
M [C , e] =C->eC
M [C , d] =C->d
```

Result: -

The program was successfully compiled and run.

Shift Reduce Parsing

Aim: Implementation of Shift Reduce Paring

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct prodn
{
        char p1[10];
        char p2[10];
};
void main()
{
        char input[20],stack[50],temp[50],ch[2],*t1,*t2,*t;
        int i,j,s1,s2,s,count=0;
        struct prodn p[10];
        FILE *fp=fopen("sr_input.txt","r");
        stack[0]='\0';
        clrscr();
        printf("\n Enter the input string\n");
        scanf("%s",&input);
        while(!feof(fp))
        {
                fscanf(fp,"%s\n",temp);
                t1=strtok(temp,"->");
                t2=strtok(NULL,"->");
                strcpy(p[count].p1,t1);
                strcpy(p[count].p2,t2);
                count++;
        }
        i=0;
        while(1)
        {
                if(i<strlen(input))
                {
                        ch[0]=input[i];
                        ch[1]='\0';
                        i++;
                        strcat(stack,ch);
                        printf("%s\n",stack);
                }
                for(j=0;j<count;j++)
                {
                        t=strstr(stack,p[j].p2);
                        if(t!=NULL)
                        {
                                s1=strlen(stack);
                                s2=strlen(t);
                                s=s1-s2;
```

```
                                stack[s]='\0';
                                strcat(stack,p[j].p1);
                                printf("%s\n",stack);
                                j=-1;
                        }
                }
                if(strcmp(stack,"E")==0&&i==strlen(input))
                {
                        printf("\n Accepted");
                        break;
                }
                if(i==strlen(input))
                {
                        printf("\n Not Accepted");
                        break;
                }
        }
        getch();
}
```

```
E->E*E
E->i
```

Output 1:

```
Enter the input string
i*i+i
 i
E
 E*
E*i
 E*E
E
 E+
 E+i
E+E
 E

 Accepted
```

Output 2:

```
Enter the input string
i*+i
 i
```

Input File: sr_input.txt

```
E->E+E
```

E
E*
E*+
E*+i
E*+E

Not Accepted

Result: Thus Implementation of Shift Reduce Parsing is successful.

LAB EXP 8 : LEADING AND TRAILING

AIM : A program to implement Leading and Trailing

ALGORITHM :

1. For Leading, check for the first non-terminal.
2. If found, print it.
3. Look for next production for the same non-terminal.
4. If not found, recursively call the procedure for the single non-terminal present before the
comma or End Of Production String.
5. Include it's results in the result of this non-terminal.
6. For trailing, we compute same as leading but we start from the end of the production to the beginning.
7. Stop

CODE :

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int vars,terms,i,j,k,m,rep,count,temp=-1;
char var[10],term[10],lead[10][10],trail[10][10];
struct grammar
{
        int prodno;
```

```
        char lhs,rhs[20][20];
}gram[50];
void get()
{
        cout<<"\n---------- LEADING AND TRAILING---------- \n";
        cout<<"\nEnter the no. of variables : ";
        cin>>vars;
        cout<<"\nEnter the variables : \n";
        for(i=0;i<vars;i++)
        {
                cin>>gram[i].lhs;
                var[i]=gram[i].lhs;
        }
        cout<<"\nEnter the no. of terminals : ";
        cin>>terms;
        cout<<"\nEnter the terminals : ";
        for(j=0;j<terms;j++)
                cin>>term[j];
        cout<<"\n---------- PRODUCTION DETAILS ---------- \n";
        for(i=0;i<vars;i++)
        {
                cout<<"\nEnter the no. of production of "<<gram[i].lhs<<":";
                cin>>gram[i].prodno;
                for(j=0;j<gram[i].prodno;j++)
                {
                        cout<<gram[i].lhs<<"->";
                        cin>>gram[i].rhs[j];
                }
        }
}
void leading()
```

```
{
        for(i=0;i<vars;i++)
        {
                for(j=0;j<gram[i].prodno;j++)
                {
                        for(k=0;k<terms;k++)
                        {
                                if(gram[i].rhs[j][0]==term[k])
                                        lead[i][k]=1;
                                else
                                {
                                        if(gram[i].rhs[j][1]==term[k])
                                                lead[i][k]=1;
                                }
                        }
                }
        }
        for(rep=0;rep<vars;rep++)
        {
                for(i=0;i<vars;i++)
                {
                        for(j=0;j<gram[i].prodno;j++)
                        {
                                for(m=1;m<vars;m++)
                                {
                                        if(gram[i].rhs[j][0]==var[m])
                                        {
                                                temp=m;
                                                goto out;
                                        }
                                }
```

```
                                out:
                                for(k=0;k<terms;k++)
                                {
                                        if(lead[temp][k]==1)
                                                lead[i][k]=1;
                                }
                        }
                }
        }
}
void trailing()
{
        for(i=0;i<vars;i++)
        {
                for(j=0;j<gram[i].prodno;j++)
                {
                        count=0;
                        while(gram[i].rhs[j][count]!='\x0')
                                count++;
                        for(k=0;k<terms;k++)
                        {
                                if(gram[i].rhs[j][count-1]==term[k])
                                        trail[i][k]=1;
                                else
                                {
                                        if(gram[i].rhs[j][count-2]==term[k])
                                                trail[i][k]=1;
                                }
                        }
                }
        }
```

```
	for(rep=0;rep<vars;rep++)
	{
		for(i=0;i<vars;i++)
		{
			for(j=0;j<gram[i].prodno;j++)
			{
				count=0;
				while(gram[i].rhs[j][count]!='\x0')
					count++;
				for(m=1;m<vars;m++)
				{
					if(gram[i].rhs[j][count-1]==var[m])
						temp=m;
				}
				for(k=0;k<terms;k++)
				{
					if(trail[temp][k]==1)
						trail[i][k]=1;
				}
			}
		}
	}
}
void display()
{
	for(i=0;i<vars;i++)
	{
		cout<<"\nLEADING("<<gram[i].lhs<<") = ";
		for(j=0;j<terms;j++)
		{
			if(lead[i][j]==1)
```

```
                                cout<<term[j]<<",";
                        }
                }
                cout<<endl;
                for(i=0;i<vars;i++)
                {
                        cout<<"\nTRAILING("<<gram[i].lhs<<") = ";
                        for(j=0;j<terms;j++)
                        {
                                if(trail[i][j]==1)
                                        cout<<term[j]<<",";
                        }
                }
}
void main()
{
        clrscr();
        get();
        leading();
        trailing();
        display();
        getch();
}
```

OUTPUT :

```
------------- LEADING AND TRAILING ---------------

Enter the no. of variables : 3

Enter the variables :
E
T
F

Enter the no. of terminals : 5

Enter the terminals : )
(
*
+
i

------------- PRODUCTION DETAILS -----------------

Enter the no. of production of E:2
E->E+T
E->T

Enter the no. of production of T:2
T->T*F
T->F

Enter the no. of production of F:2
F->(E)
F->i

LEADING(E) = (,*,+,i,
LEADING(T) = (,*,i,
LEADING(F) = (,i,

TRAILING(E) = ),*,+,i,
TRAILING(T) = ),*,i,
TRAILING(F) = ),i,
```

RESULT :

The program was successfully compiled and run.

Computation of LR(0) Items

Aim: A program to implement LR(0) items

Algorithm:-

1. Start.
2. Create structure for production with LHS and RHS.
3. Open file and read input from file.
4. Build state 0 from extra grammar Law S' -> S $ that is all start symbol of grammar and one Dot (.) before S symbol.
5. If Dot symbol is before a non-terminal, add grammar laws that this non-terminal is in Left Hand Side of that Law and set Dot in before of first part of Right Hand Side.
6. If state exists (a state with this Laws and same Dot position), use that instead.
7. Now find set of terminals and non-terminals in which Dot exist in before.
8. If step 7 Set is non-empty go to 9, else go to 10.
9. For each terminal/non-terminal in set step 7 create new state by using all grammar law that Dot position is before of that terminal/non-terminal in reference state by increasing Dot point to next part in Right Hand Side of that laws.
10. Go to step 5.
11. End of state building.
12. Display the output.
13. End.

Program:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
char prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQR";
int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
```

```
int noitem=0;
struct Grammar
{
	char lhs;
	char rhs[8];
}g[20],item[20],clos[20][10];

int isvariable(char variable)
{
	for(int i=0;i<novar;i++)
		if(g[i].lhs==variable)
			return i+1;
	return 0;
}
void findclosure(int z, char a)
{
	int n=0,i=0,j=0,k=0,l=0;
	for(i=0;i<arr[z];i++)
	{
		for(j=0;j<strlen(clos[z][i].rhs);j++)
		{
			if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==a)
			{
				clos[noitem][n].lhs=clos[z][i].lhs;
				strcpy(clos[noitem][n].rhs,clos[z][i].rhs);
				char temp=clos[noitem][n].rhs[j];
				clos[noitem][n].rhs[j]=clos[noitem][n].rhs[j+1];
				clos[noitem][n].rhs[j+1]=temp;
				n=n+1;
			}
		}
```

```
	}
	for(i=0;i<n;i++)
	{
		for(j=0;j<strlen(clos[noitem][i].rhs);j++)
		{
			if(clos[noitem][i].rhs[j]=='.' && isvariable(clos[noitem][i].rhs[j+1])>0)
			{
				for(k=0;k<novar;k++)
				{
					if(clos[noitem][i].rhs[j+1]==clos[0][k].lhs)
					{
						for(l=0;l<n;l++)
							if(clos[noitem][l].lhs==clos[0][k].lhs	&&
strcmp(clos[noitem][l].rhs,clos[0][k].rhs)==0)
								break;
						if(l==n)
						{
							clos[noitem][n].lhs=clos[0][k].lhs;
						strcpy(clos[noitem][n].rhs,clos[0][k].rhs);
							n=n+1;
						}
					}
				}
			}
		}
	}
	arr[noitem]=n;
	int flag=0;
	for(i=0;i<noitem;i++)
	{
		if(arr[i]==n)
```

```
		{
			for(j=0;j<arr[i];j++)
			{
				int c=0;
				for(k=0;k<arr[i];k++)
					if(clos[noitem][k].lhs==clos[i][k].lhs &&
strcmp(clos[noitem][k].rhs,clos[i][k].rhs)==0)
						c=c+1;
				if(c==arr[i])
				{
					flag=1;
					goto exit;
				}
			}
		}
	}
	exit:;
	if(flag==0)
		arr[noitem++]=n;
}

void main()
{
	clrscr();
	cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n";
	do
	{
		cin>>prod[i++];
	}while(strcmp(prod[i-1],"0")!=0);
	for(n=0;n<i-1;n++)
	{
```

```
        m=0;
        j=novar;
        g[novar++].lhs=prod[n][0];
        for(k=3;k<strlen(prod[n]);k++)
        {
                if(prod[n][k] != '|')
                g[j].rhs[m++]=prod[n][k];
                if(prod[n][k]=='|')
                {
                        g[j].rhs[m]='\0';
                        m=0;
                        j=novar;
                        g[novar++].lhs=prod[n][0];
                }
        }
}
for(i=0;i<26;i++)
        if(!isvariable(listofvar[i]))
                break;
g[0].lhs=listofvar[i];
char temp[2]={g[1].lhs,'\0'};
strcat(g[0].rhs,temp);
cout<<"\n\n augumented grammar \n";
for(i=0;i<novar;i++)
        cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";
getch();
for(i=0;i<novar;i++)
{
        clos[noitem][i].lhs=g[i].lhs;
        strcpy(clos[noitem][i].rhs,g[i].rhs);
        if(strcmp(clos[noitem][i].rhs,"ε")==0)
```

```
                strcpy(clos[noitem][i].rhs,".");
        else
        {
                for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)
                        clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1];
                clos[noitem][i].rhs[0]='.';
        }
}
arr[noitem++]=novar;
for(int z=0;z<noitem;z++)
{
        char list[10];
        int l=0;
        for(j=0;j<arr[z];j++)
        {
                for(k=0;k<strlen(clos[z][j].rhs)-1;k++)
                {
                        if(clos[z][j].rhs[k]=='.')
                        {
                                for(m=0;m<l;m++)
                                        if(list[m]==clos[z][j].rhs[k+1])
                                                break;
                                if(m==l)
                                        list[l++]=clos[z][j].rhs[k+1];
                        }
                }
        }
        for(int x=0;x<l;x++)
                findclosure(z,list[x]);
}
cout<<"\n THE SET OF ITEMS ARE \n\n";
```

```
        for(z=0;z<noitem;z++)
        {
                cout<<"\n I"<<z<<"\n\n";
                for(j=0;j<arr[z];j++)
                        cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";
                getch();
        }
        getch();
}
```

Output:-

```
ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
0
 augumented grammar

A->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
 THE SET OF ITEMS ARE
 I0
A->.E
E->.E+T
```

```
E->.T
T->.T*F
T->.F
F->.(E)
F->.i

 I1

A->E.
E->E.+T

 I2

E->T.
T->T.*F

 I3

T->F.

 I4

F->(.E)
E->.E+T
E->.T
T->.T*F
T->.F
F->.(E)
F->.i

 I5
F->i.

 I6
```

```
E->E+.T
T->.T*F
T->.F
F->.(E)
F->.i

 I7

T->T*.F
F->.(E)
F->.i

 I8

F->(E.)
E->E.+T

 I9

E->E+T.
T->T.*F

 I10

T->T*F.

 I11

F->(E).
```

Result:-

The program was successfully compiled and run.

Intermediate code generation – Postfix, Prefix

Aim: A program to implement Intermediate code generation – Postfix, Prefix.

Algorithm:-

1. Declare set of operators.
2. Initialize an empty stack.
3. To convert INFIX to POSTFIX follow the following steps
4. Scan the infix expression from left to right.
5. If the scanned character is an operand, output it.
6. Else, If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.
7. Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack.
8. If the scanned character is an '(', push it to the stack.
9. If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
10. Pop and output from the stack until it is not empty.
11. To convert INFIX to PREFIX follow the following steps
12. First, reverse the infix expression given in the problem.
13. Scan the expression from left to right.
14. Whenever the operands arrive, print them.
15. If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
16. Repeat steps 6 to 9 until the stack is empty

Program:

```
OPERATORS = set(['+', '-', '*', '/', '(', ')'])

PRI = {'+': 1, '-': 1, '*': 2, '/': 2}


### INFIX ===> POSTFIX ###

def infix_to_postfix(formula):
    stack = [] # only pop when the coming op has priority

    output = ''

    for ch in formula:

        if ch not in OPERATORS:

            output += ch

        elif ch == '(':

            stack.append('(')

        elif ch == ')':

            while stack and stack[-1] != '(':
                output += stack.pop()

            stack.pop() # pop '('

        else:
```

```
            while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
                output += stack.pop()

            stack.append(ch)

            # leftover

    while stack:
        output += stack.pop()

    print(f'POSTFIX: {output}')

    return output


### INFIX ===> PREFIX ###

def infix_to_prefix(formula):
    op_stack = []

    exp_stack =  []

    for ch in formula:

        if not ch in OPERATORS:

            exp_stack.append(ch)

        elif ch == '(':
```

```
            op_stack.append(ch)

        elif ch == ')':

            while op_stack[-1] != '(':
                op = op_stack.pop()

                a = exp_stack.pop()

                b = exp_stack.pop()

                exp_stack.append(op + b + a)

            op_stack.pop() # pop '('

        else:

            while op_stack and op_stack[-1] != '(' and PRI[ch] <= PRI[op_stack[-1]]:
                op = op_stack.pop()

                a = exp_stack.pop()

                b = exp_stack.pop()

                exp_stack.append(op + b + a)

            op_stack.append(ch)

            # leftover

    while op_stack:
```

```
        op = op_stack.pop()
        a = exp_stack.pop()
        b = exp_stack.pop()
        exp_stack.append(op + b + a)
    print(f'PREFIX: {exp_stack[-1]}')
    return exp_stack[-1]
expres = input("INPUT THE EXPRESSION: ")
pre = infix_to_prefix(expres)
pos = infix_to_postfix(expres)
```

Output:-

```
PS E:\Studies\SRM University\SEM 6\Complier Design> python
INPUT THE EXPRESSION: A+B^C/R
PREFIX: +^/CR
POSTFIX: AB^CR/+
```

Result:-

The program was successfully compiled and run.

Intermediate code generation – Quadruple, Triple, Indirect triple

Aim: Intermediate code generation – Quadruple, Triple, Indirect triple

Algorithm:-

The algorithm takes a sequence of three-address statements as input. For each three address statements of the form a:= b op c perform the various actions. These are as follows: 1. Invoke a function getreg to find out the location L where the result of computation b op c should be stored.

2. Consult the address description for y to determine y'. If the value of y currently in memory and register both then prefer the register y' . If the value of y is not already in L then generate the instruction MOV y' , L to place a copy of y in L.
3. Generate the instruction OP z' , L where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptors.
4. If the current value of y or z have no next uses or not live on exit from the block or in register then alter the register descriptor to indicate that after execution of x : = y op z those register will no longer contain y or z.

Program:

```
#include<stdio.h>
#include<ctype.h>
```

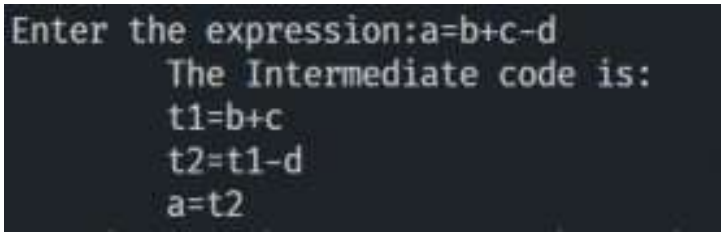
```
#include<stdlib.h>
#include<string.h>
void small();
void dove(int i);
int p[5]={0,1,2,3,4},c=1,i,k,l,m,pi;
char sw[5]={'=','-','+','/','*'},j[20],a[5],b[5],ch[2];
void main()
{
printf("Enter the expression:");
scanf("%s",j);
printf("\tThe Intermediate code is:\n");
small();
}
void dove(int i)
{
a[0]=b[0]='\0';
if(!isdigit(j[i+2])&&!isdigit(j[i-2]))
{
a[0]=j[i-1];
b[0]=j[i+1];
}
if(isdigit(j[i+2])){
a[0]=j[i-1];
```

```
b[0]='t';
b[1]=j[i+2];
 }
 if(isdigit(j[i-2]))
 {
b[0]=j[i+1];
 a[0]='t';
 a[1]=j[i-2];
b[1]='\0';
 }
 if(isdigit(j[i+2]) &&isdigit(j[i-2]))
 {
 a[0]='t';
b[0]='t';
 a[1]=j[i-2];
b[1]=j[i+2];
 sprintf(ch,"%d",c);
j[i+2]=j[i-2]=ch[0];
 }
 if(j[i]=='*')
printf("\tt%d=%s*%s\n",c,a,b);
if(j[i]=='/')
```

```
printf("\tt%d=%s/%s\n",c,a,b);
if(j[i]=='+')
printf("\tt%d=%s+%s\n",c,a,b);if(j[i]=='-')
printf("\tt%d=%s-%s\n",c,a,b);
if(j[i]=='=')
printf("\t%c=t%d",j[i-1],--c);
sprintf(ch,"%d",c);
j[i]=ch[0];
c++;
small();
}
void small()
{
pi=0;l=0;
for(i=0;i<strlen(j);i++)
{
for(m=0;m<5;m++)
if(j[i]==sw[m])
if(pi<=p[m])
{
pi=p[m];
l=1;
```

```
k=i;
}
}
if(l==1)
dove(k);
else
exit(0);}
```

Output:-



Result:-

The program was successfully compiled and run.

Simple Code Generator

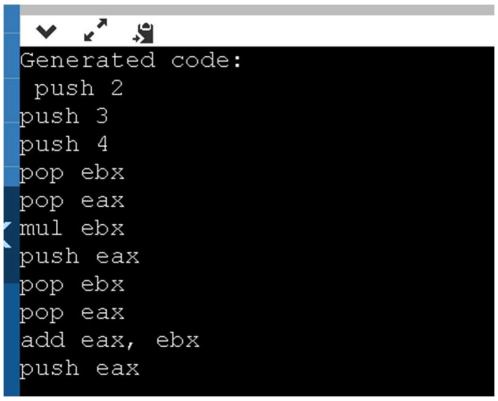
Aim: Implementation of Simple code Generator.

Program:

```
def generate_code(expr):
    if expr.type == 'number':
        return f"push {expr.value}\n"
    elif expr.type == 'add':
        code = ""
        code += generate_code(expr.left)
        code += generate_code(expr.right)
        code += "pop ebx\npop eax\nadd eax, ebx\npush eax\n"
        return code
    elif expr.type == 'subtract':
        code = ""
        code += generate_code(expr.left)
        code += generate_code(expr.right)
        code += "pop ebx\npop eax\nsub eax, ebx\npush eax\n"
        return code
    elif expr.type == 'multiply':
        code = ""
        code += generate_code(expr.left)
        code += generate_code(expr.right)
        code += "pop ebx\npop eax\nmul ebx\npush eax\n"
        return code
    elif expr.type == 'divide':
        code = ""
        code += generate_code(expr.left)
        code += generate_code(expr.right)
        code += "pop ebx\npop eax\nmov edx, 0\ndiv ebx\npush eax\n"
        return code

class Number:
    def __init__(self, value):
        self.type = 'number'
        self.value = value

class Add:
    def __init__(self, left, right):
```

```
        self.type = 'add'
        self.left = left
        self.right = right

class Subtract:
    def __init__(self, left, right):
        self.type = 'subtract'
        self.left = left
        self.right = right

class Multiply:
    def __init__(self, left, right):
        self.type = 'multiply'
        self.left = left
        self.right = right

class Divide:
    def __init__(self, left, right):
        self.type = 'divide'
        self.left = left
        self.right = right

expr = Add(Number(2), Multiply(Number(3), Number(4)))
code = generate_code(expr)
print("Generated code:\n", code)
```

output:



Result: Thus Implementation of Simple Code Generator is successful.

Experiment No: 13

EXPERIMENT TITLE: Implementation of DAG

Aim: A program to implement DAG

Algorithm:

1. The leaves of a graph are labeled by a unique identifier and that identifier can be variable names or constants.
2. Interior nodes of the graph are labeled by an operator symbol.
3. Nodes are also given a sequence of identifiers for labels to store the computed value.
4. If y operand is undefined then create node(y).
5. If z operand is undefined then for case(i) create node(z).
6. For case(i), create node(OP) whose right child is node(z) and left child is node(y).
7. For case(ii), check whether there is node(OP) with one child node(y).
8. For case(iii), node n will be node(y).
9. For node(x) delete x from the list of identifiers. Append x to attached identifiers list for the node n found in step 2. Finally set node(x) to n.

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
void small();
void dove(int i);
int p[5]={0,1,2,3,4},c=1,i,k,l,m,pi;
char sw[5]={'=','-','+','/','*'},j[20],a[5],b[5],ch[2];
void main()
{
```

```
printf("Enter the expression:");
scanf("%s",j);
printf("\tThe Intermediate code is:\n");
small();
}
void dove(int i)
{
a[0]=b[0]='\0';
if(!isdigit(j[i+2])&&!isdigit(j[i-2]))
{
a[0]=j[i-1];
b[0]=j[i+1];
}
if(isdigit(j[i+2])){
a[0]=j[i-1];
b[0]='t';
b[1]=j[i+2];
}
if(isdigit(j[i-2]))
{
b[0]=j[i+1];
a[0]='t';
a[1]=j[i-2];
b[1]='\0';
}
if(isdigit(j[i+2]) &&isdigit(j[i-2]))
{
a[0]='t';
b[0]='t';
a[1]=j[i-2];
b[1]=j[i+2];
sprintf(ch,"%d",c);
j[i+2]=j[i-2]=ch[0];
}
if(j[i]=='*')
printf("\tt%d=%s*%s\n",c,a,b);
if(j[i]=='/')
printf("\tt%d=%s/%s\n",c,a,b);
```

```
if(j[i]=='+')
printf("\tt%d=%s+%s\n",c,a,b);if(j[i]=='-')
printf("\tt%d=%s-%s\n",c,a,b);
if(j[i]=='=')
printf("\t%c=t%d",j[i-1],--c);
sprintf(ch,"%d",c);
j[i]=ch[0];
c++;
small();
}
void small()
{
pi=0;l=0;
for(i=0;i<strlen(j);i++)
{
for(m=0;m<5;m++)
if(j[i]==sw[m])
if(pi<=p[m])
{
pi=p[m];
l=1;
k=i;
if(l==1)
dove(k);
else
exit(0);}
```

Output :

```
Macbook-Air:desktop ish$ ./a.out
Enter the expression:a=b+c-d
        The Intermediate code is:
        t1=b+c
        t2=t1-d
Macbook-Air:desktop ish$
```

Result: The program was successfully compiled and run.

ID: 08BC482B0E6F

HackerRank

Certificate

This is to certify that

EKKULURI RAJESH

has successfully cleared the assessment for the skill

Java (Basic)

12 May, 2023

Date

Harishankaran K

CTO, HackerRank