PORTFOLIO

COMPILER DESIGN

SHREYA SHUKLA

RA1911003010429

LAB EXERCISES

Shreya Shukla
RA1911003010429

Compiler Design

Exp 1 - Lexical Analyzer

Aim - To construct a lexical analyzer to convert the given input.

Procedure -

1) Get the input program from the file program.txt
2) Read the program line by line and check if each word is a line, keyword, identifier, constant or an operator.
3) Identify the different characters & print them on the output screen accordingly.

PROGRAM -

```
file = open ("program.txt", "r")
lines = file.readlines()
keywords = ["void", "main", "int", "float", "bool",
            "if", "for", "else", "while", "char",
            "return"]

operators = ["=", "==", "+", "-", "*", "/",
             "++", "--", "+=", "!=", "-=",
             "||", "&&"]

punctuations = [";", ",", ".", ")", "(",
                "{", "}", "[", "]"]
```

```
def isinteger(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 isinteger(n):
            print(i, "is a number")

        else
            print(i, "is an identifier)
```

INPUT: (program.txt)

```
int main(){
    int a, b, c;
    c = a+b;
}
```

Shreya Shukla, 429

OUTPUT -

int is a keyword
main is a keyword
(is a punctuation
) is a punctuation
int is a keyword
a is an identifier
, is a punctuation
b is an identifier
, is a punctuation
c is an identifier
; is a punctuation
= is an operator
+ is an operator
b is an identifier
; is a punctuation
} is a punctuation

RESULT - The implementation of lexical analyzer in python was compiled, executed & verified successfully.

EXERCISE 2:

CONVERSION OF REGULAR EXPRESSION TO NFA

SHREYA SHUKLA
RA1911003010429

AIM: To convert a given regular expression into its non-deterministic finite automata representation using C++.

PROCEDURE:

STEP 1: A structure is created for each kind of regular expression that we want to create into NDFA.
STEP 2: Under each structure, we define 3 variables - start state, alphabet input and n-state.
STEP 3: Each state changes into another state based on the alphabet input symbol. All these are defined in the structures
STEP 4: A switch case is made with 4 choices (4 different regular expressions) and based on the choice selected, an NFA diagram is shown as output.

PROGRAM/CODE:

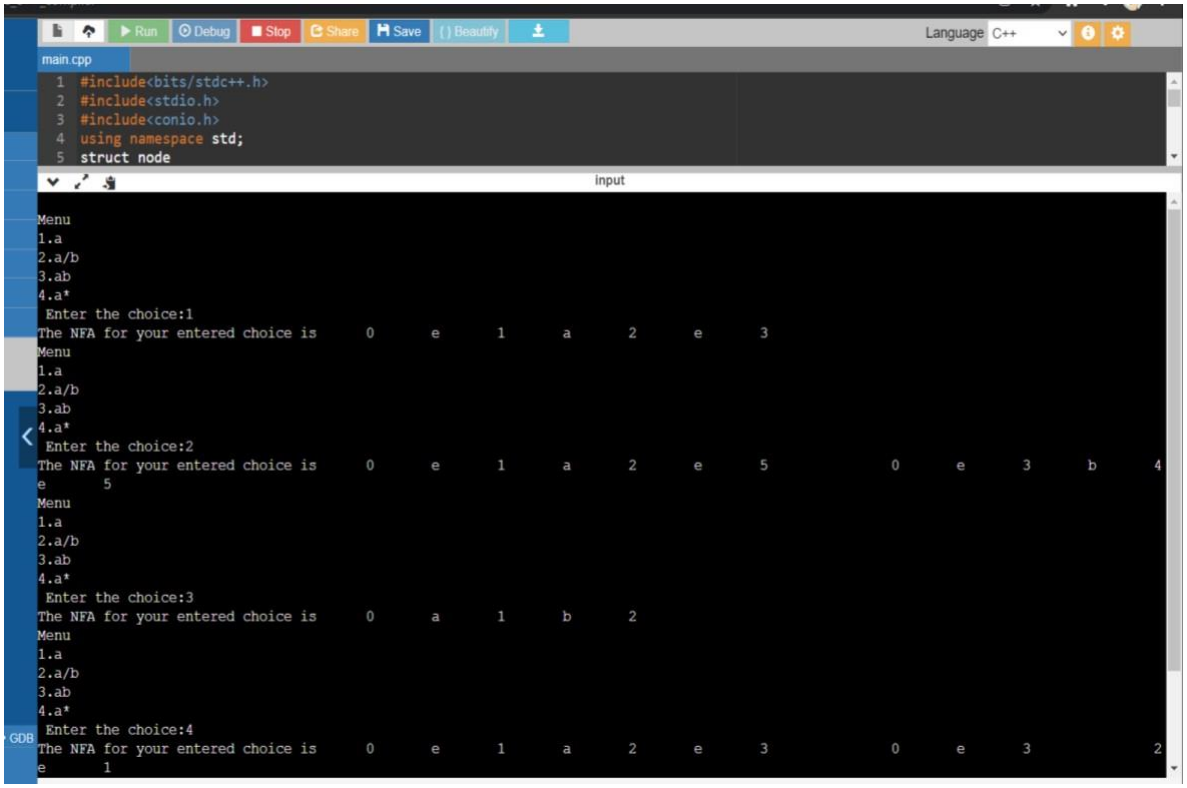
```
#include<bits/stdc++.h>
#include<stdio.h>
#include<conio.h>
using namespace std;
struct node
{ char start;
char alp;
node
*nstate;
}*p,*p1,*p2,*p3,*p4,*p5,*p6,*p7,*p8
; char e='e'; void disp(); void re1()
{
p1=new(node);
p2=new(node);
p3=new(node);
p4=new(node);
p1->start='0';
p1->alp='e'; p1-
>nstate=p2; p2-
>start='1'; p2-
>alp='a'; p2-
>nstate=p3; p3-
>start='2'; p3-
>alp='e'; p3-
>nstate=p4;
p4->start='3';
p4->nstate=NULL;
```

```
disp();
getch(); }
void
re2()
{
p1=new(node);
p2=new(node);
p3=new(node);
p4=new(node);
p5=new(node);
p6=new(node);
p7=new(node);
p8=new(node);
p1->start='0';
p1->alp='e'; p1-
>nstate=p2; p2-
>start='1'; p2-
>alp='a'; p2-
>nstate=p3; p3-
>start='2'; p3-
>alp='e'; p3-
>nstate=p4; p4-
>start='5'; p4-
>alp=' '; p4-
>nstate=p5; p5-
>start='0'; p5-
>alp='e'; p5-
>nstate=p6; p6-
>start='3'; p6-
>alp='b'; p6-
>nstate=p7; p7-
>start='4'; p7-
>alp='e'; p7-
>nstate=p8; p8-
>start='5';
p8->alp=' ';
p8-
>nstate=NULL;
disp(); getch(); }
void re3()
{
p1=new(node);
p2=new(node);
p3=new(node);
p1->start='0';
p1->alp='a'; p1-
>nstate=p2; p2-
>start='1'; p2-
>alp='b'; p2-
```

```
>nstate=p3; p3-
>start='2'; p3-
>alp=' ';
p3->nstate=NULL;
disp();
getch(); }
void
re4()
{
p1=new(node);
p2=new(node);
p3=new(node);
p4=new(node);
p5=new(node);
p6=new(node);
p7=new(node);
p8=new(node);
p1->start='0';
p1->alp='e'; p1-
>nstate=p2; p2-
>start='1'; p2-
>alp='a'; p2-
>nstate=p3; p3-
>start='2'; p3-
>alp='e'; p3-
>nstate=p4; p4-
>start='3'; p4-
>alp=' '; p4-
>nstate=p5; p5-
>start='0'; p5-
>alp='e'; p5-
>nstate=p6; p6-
>start='3'; p6-
>alp=' '; p6-
>nstate=p7; p7-
>start='2'; p7-
>alp='e'; p7-
>nstate=p8; p8-
>start='1';
p8->alp=' ';
p8->nstate=NULL;
disp();
getch(); }
void
disp() {
p=p1;
while(p!=NULL)
{ cout<<"\t"<<p->start<<"\t"<<p-
>alp; p=p->nstate;
```

```
}
}
int
m
ai
n(
)
{
p=new(node);
int ch=1;
while(ch!=0)
{
cout<<"\nMenu"<<"\n1.a"<<"\n2.a/b"<<"\n3.ab"<<"\n4.a*";
cout<<"\n Enter the
choice:"; cin>>ch;
switch(ch) { case 1: { re1();
break; } case 2: { re2();
break; } case 3: { re3();
break; } case 4: { re4();
break; } default: { exit(0); }
}
} return
0;
}
```

OUTPUT:



RESULT: Thus, we have successfully converted the given Regular expression into NFA using C++.

Exercise 3- Non-Deterministic Finite Automata to Deterministic Finite Automata

SHREYA SHUKLA
RA1911003010429

AIM: Write a program in your preferred language to convert a Non-Deterministic Finite Automata to a Deterministic Finite Automata.

ALGORITHM:

Step 1: Accept the list of states, input alphabets (sigma) and the NFA transitions from the user.

Step 2: Create a dictionary that contains the NFA transitions for each state based on the given user inputs.

Step 3: Display the dictionary items (from step 2) to show the transition table of NFA.

Step 4: Create an NFA using the automata.fa Python package, where the states, input symbols, transitions, the initial state and final states are assigned.

Step 5: Validate the NFA (from step 4) using validate().

Step 6: Accept an input string from the user and check if it is accepted or rejected by the given NFA using accepts_input().

Step 7: Convert the NFA to its equivalent DFA using the from_nfa() method.

Step 8: Repeat steps 5 and 6 for the obtained DFA and displays its transition table.

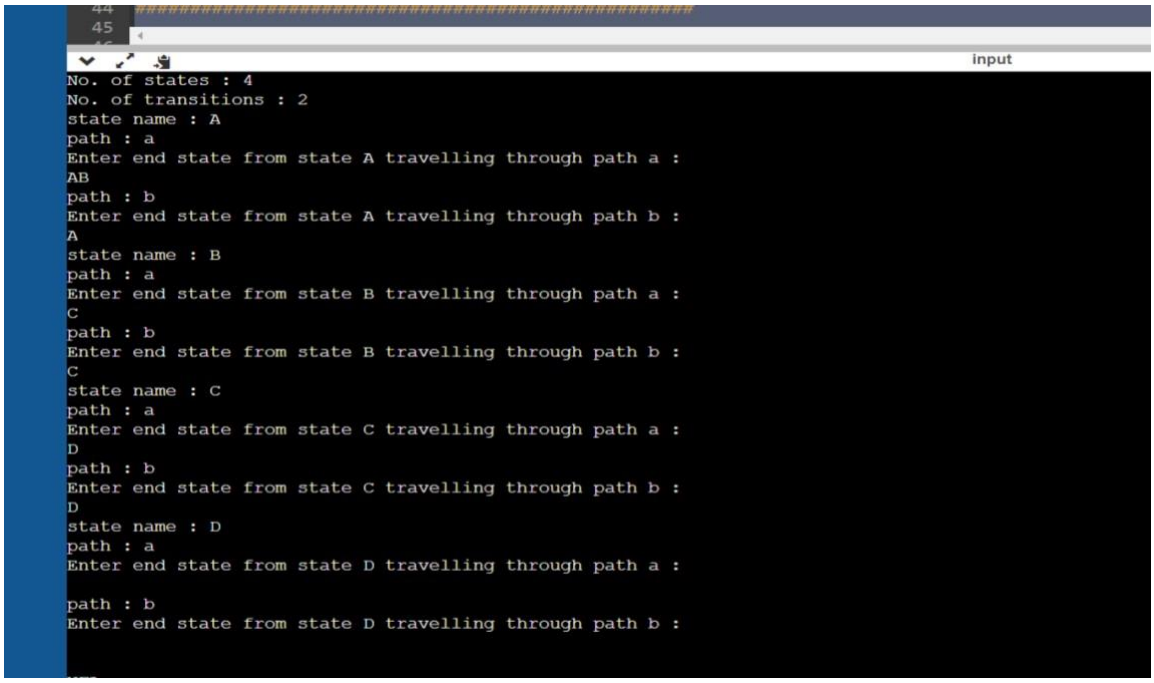
CODE:

```
import pandas as pd

# Taking NFA input from User

nfa = {}
n = int(input("No. of states : "))          #Enter total no. of states t = int(input("No. of
transitions : "))      #Enter total no. of transitions/paths eg: a,b so input 2 for a,b,c input 3
for i in range(n):       state = input("state name : ")          #Enter state name eg: A, B, C,
q1, q2 ..etc     nfa[state] = {}                        #Creating a nested dictionary     for j in
range(t):        path = input("path : ")             #Enter path eg : a or b in {a,b} 0 or 1 in
{0,1}        print("Enter end state from state {} travelling through path {} :
".format(state,path))        reaching_state = [x for x in input().split()]  #Enter all the end
states that
      nfa[state][path] = reaching_state     #Assigning the end states to the paths in dictionary

print("\nNFA :- \n")
print(nfa)                              #Printing 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()]      # Enter
final state/states of NFA new_states_list = []                        #holds all the new states
created in dfa dfa = {}                                  #dfa dictionary/table or the output
structure we needed keys_list = list(list(nfa.keys())[0])                #conatins all the states
in nfa plus the states
created in dfa are also appended further
path_list = list(nfa[keys_list[0]].keys())    #list of all the paths eg: [a,b] or [0,1]


# Computing first row of DFA transition table

dfa[keys_list[0]] = {}                       #creating a nested dictionary in dfa  for y in
range(t):     var = "".join(nfa[keys_list[0]][path_list[y]])   #creating a single string
from all the
elements of the list which is a new state
   dfa[keys_list[0]][path_list[y]] = var           #assigning the state in DFA table     if var
not in keys_list:                       #if the state is newly created
new_states_list.append(var)                #then append it to the new_states_list
keys_list.append(var)                     #as well as to the keys_list which contains all the
states

# Computing the other rows of DFA transition table

while len(new_states_list) != 0:                    #consition is true only if the new_states_list is
not empty
   dfa[new_states_list[0]] = {}                   #taking the first element of the new_states_list
and examining it
   for _ in range(len(new_states_list[0])):
for i in range(len(path_list)):
         temp = []                            #creating a temporay list
for j in range(len(new_states_list[0])):
            temp += nfa[new_states_list[0][j]][path_list[i]]  #taking the union of the
states           s = ""           s = s.join(temp)                      #creating a single string(new
state) from all the
elements of the list
         if s not in keys_list:                  #if the state is newly created
            new_states_list.append(s)           #then append it to the new_states_list
keys_list.append(s)                #as well as to the keys_list which contains all the
states
         dfa[new_states_list[0]][path_list[i]] = s   #assigning the new state in the DFA table

   new_states_list.remove(new_states_list[0])       #Removing the first element in the
```

```
new_states_list

print("\nDFA :- \n")
print(dfa)                                  #Printing the DFA created
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)       #Printing Final states of DFA
```

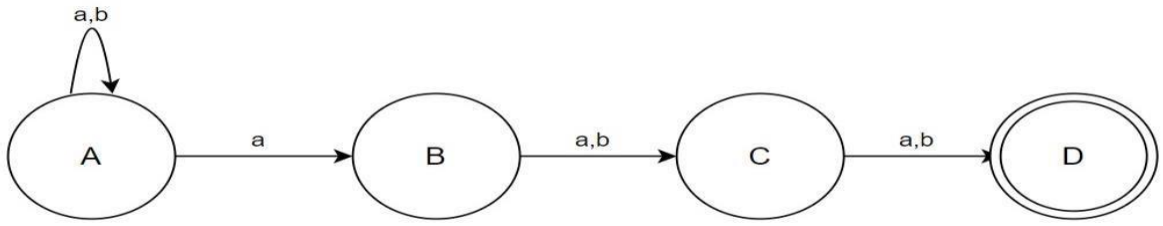
OUTPUT:



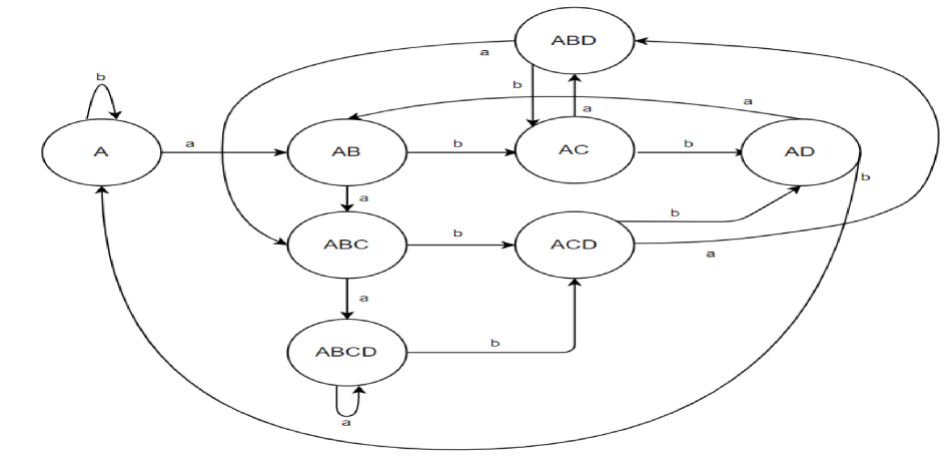
```
NFA :-

{'A': {'a': ['AB'], 'b': ['A']}, 'B': {'a': ['C'], 'b': ['C']}, 'C': {'a': ['D'], 'b': ['D']}, 'D': {'a': [], 'b': []}}

Printing NFA table :-
      a    b
A  [AB]  [A]
B   [C]  [C]
C   [D]  [D]
D    []   []
Enter final state of NFA :
{'A':{'a':['A','B'],'b':['A']},'B':{'a':['C'],'b':['C']},'C':{'a':['D'],'b':['D']},'D':{'a':[],'b':[]}}

DFA :-

{'A': {'a': 'AB', 'b': 'A'}, 'AB': {'a': 'ABC', 'b': 'AC'}, 'ABC': {'a': 'ABCD', 'b': 'ACD'}, 'AC': {'a': 'ABD', 'b': 'AD'}, 'ABCD': {'a'
: 'ABCD', 'b': 'ACD'}, 'ACD': {'a': 'ABD', 'b': 'AD'}, 'ABD': {'a': 'ABC', 'b': 'AC'}, 'AD': {'a': 'AB', 'b': 'A'}}

Printing DFA table :-
         a    b
A       AB    A
AB     ABC   AC
ABC   ABCD  ACD
AC     ABD   AD
ABCD  ABCD  ACD
ACD    ABD   AD
ABD    ABC   AC
AD      AB    A
```

Diagrams :

NFA : input



DFA : output



RESULT:

A program was written down in python to convert a Non-Deterministic Finite Automata to a Deterministic Finite Automata.

SHREYA SHUKLA, RA1911003010429

EXP 4 -ELIMINATION OF LEFT RECURSION AND LEFT FACTORING

AIM:

To create a program which identifies the given grammar as left recursive or not and then perform elimination of the left recursion and left factoring

REQUIREMENTS:

1. Knowledge of the concepts left and right recursive grammar
2. Knowledge of the methodology to eliminate left recursion and left factoring
3. Online GDB compiler to execute and implement code

THEORY:

Left Recursive Grammar:

- A production of grammar is said to have left recursion if the leftmost variable of its RHS is the same as the variable of its LHS.
- A grammar containing a production having left recursion is called as Left Recursive Grammar
- Left recursion is considered to be a problematic situation for Top-down parsers.
- Therefore, left recursion has to be eliminated from the grammar.

Elimination of Left Recursion

Left recursion is eliminated by converting the grammar into a right recursive grammar.

If we have the left-recursive pair of productions-

$$A \rightarrow A\alpha / \beta$$

(Left Recursive Grammar) where

β does not begin with an A.

Then, we can eliminate left recursion by replacing the pair of productions with-

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A' / \in$$

(Right Recursive Grammar)

This right recursive grammar functions the same as left recursive grammar.

Left Factoring:

Left factoring is used so that it makes the grammar useful for top-down parsers. Here we take out the left factor that appears in 2 productions of the same non-terminal. So, it is done to avoid backtracking by the parser.

ALGORITHM:

1. We declare d,a,b strings and a flag bit
2. Store the parent non-terminal in c var
3. Push c into d string such that E-> left production
4. Take the no of productions in. For example, take 2
5. And for loop till iterator = 2
6. And take each production and add it to d string var such that after each production it will be E->E+T|

To detect if the production is left recursive or not:

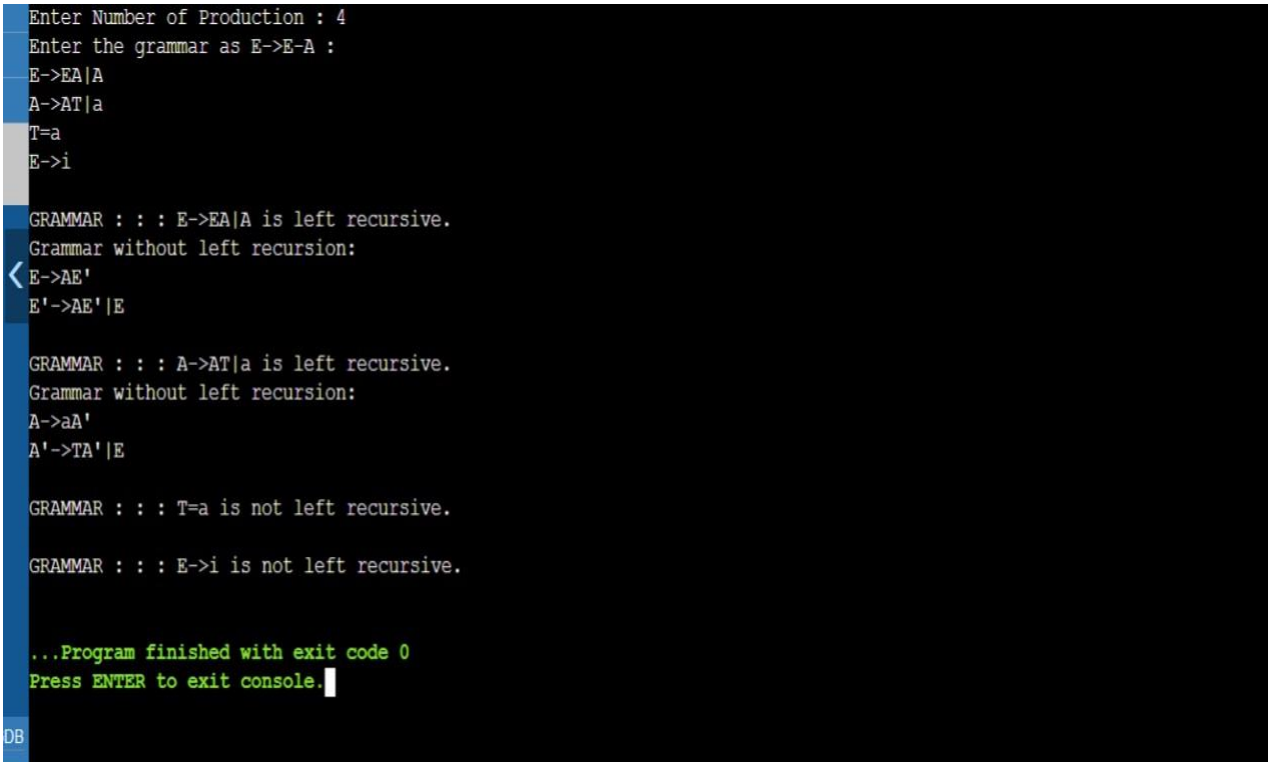
7. If(d[0] != d[k]) it doesn't have left recursion because E->E here k is the last var E and for left recursion the left variable should be the parent non-terminal.
8. If they have left recursion then push it to A vector array according to the laws of left recursion ie TE' where E'->+TE'|# AND E->TE'
9. Print the two.

CODE FOR LEFT RECURSION:

```
#include<stdio.h>
#include<string.h>
#define SIZE 10     int
main () {          char
non_terminal;
char beta,alpha;
      int num;
      char production[10][SIZE];
      int index=3; /* starting of the string following "->" */
printf("Enter Number of Production : ");          scanf("%d",&num);
      printf("Enter the grammar as E->E-A :\n");
      for(int i=0;i<num;i++){
          scanf("%s",production[i]);
      }
      for(int i=0;i<num;i++){
          printf("\nGRAMMAR : : : %s",production[i]);
non_terminal=production[i][0];
if(non_terminal==production[i][index]) {
alpha=production[i][index+1];                    printf(" is
left recursive.\n");
              while(production[i][index]!=0 && production[i][index]!='|')
index++;
              if(production[i][index]!=0) {
beta=production[i][index+1];
                   printf("Grammar without left recursion:\n");
printf("%c->%c%c\'",non_terminal,beta,non_terminal);
printf("\n%c\'-
>%c%c\'|E\n",non_terminal,alpha,non_terminal);
```

```
            }
else
printf(" can't be
reduced\n");
          }
else
            printf(" is not left recursive.\n");
index=3;
     }
 }
```

OUTPUT:



CODE FOR LEFT FACTORING:

```
#include<stdio.h>
#include<string.h>
  int main()
  {
char
gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20
];
```

```
    int i,j=0,k=0,l=0,pos;
printf("Enter Production : A->");
gets(gram);
for(i=0;gram[i]!='|';i++,j++)
part1[j]=gram[i];          part1[j]='\0';
    for(j=++i,i=0;gram[j]!='\0';j++,i++)
        part2[i]=gram[j];
part2[i]='\0';
    for(i=0;i<strlen(part1)||i<strlen(part2);i++)
    {
        if(part1[i]==part2[i])
        {
            modifiedGram[k]=part1[i];
            k++;
pos=i+1;
        }
    }
    for(i=pos,j=0;part1[i]!='\0';i++,j++){
newGram[j]=part1[i];
    }
    newGram[j++]='|';
for(i=pos;part2[i]!='\0';i++,j++){
        newGram[j]=part2[i];
    }
    modifiedGram[k]='X';
modifiedGram[++k]='\0';          newGram[j]='\0';
    printf("\n A->%s",modifiedGram);
printf("\n X->%s\n",newGram);
 }
```

OUTPUT:

OBSERVATION:

The grammars given as input were accurately detected as left recursive or not. Those identified as left recursive underwent left recursion elimination and left factoring elimination. The output was verified.

RESULT:

Thus, we have successfully implemented a program to detect and eliminate left recursive grammar.

SHREYA SHUKLA
RA1911003010429

FIRST AND FOLLOW

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

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 FOLLOW set.

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

CODE FOR FIRST:

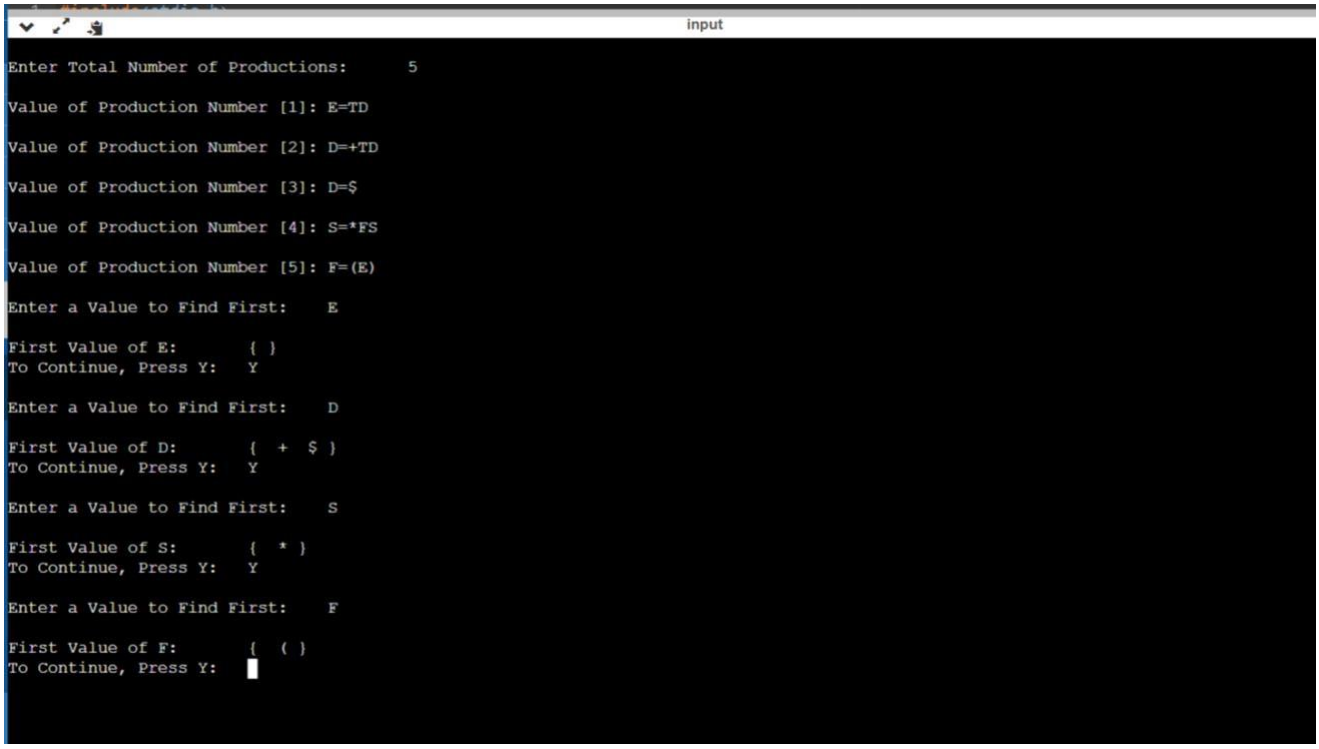
```
#include<stdio.h>
#include<ctype.h>

void Find_First(char[], char);
void Array_Manipulation(char[], char);

int limit;
char production[25][25];

int main()
{
    char option;
char ch;        char
array[25];        int
count;
```

```
    printf("\nEnter Total Number of Productions:\t");
scanf("%d", &limit);
    for(count = 0; count < limit; count++)
    {
        printf("\nValue of Production Number [%d]:\t", count + 1);
scanf("%s", production[count]);
    }
do
    {
        printf("\nEnter a Value to Find First:\t");
        scanf("              %c",              &ch);
Find_First(array, ch);            printf("\nFirst
Value of %c:\t{ ", ch);
        for(count = 0; array[count] != '\0'; count++)
        {
            printf(" %c ", array[count]);
        }
        printf("}\n");
        printf("To Continue, Press Y:\t");
scanf(" %c", &option);
    }while(option == 'y' || option == 'Y');
    return 0;
}

void Find_First(char* array, char ch)
{
    int count, j, k;      char
temporary_result[20];
    int x;
    temporary_result[0] = '\0';
    array[0] = '\0';
if(!(isupper(ch)))
    {
        Array_Manipulation(array, ch);
return ;
    }
    for(count = 0; count < limit; count++)
    {
```

```
        if(production[count][0] == ch)
        {
            if(production[count][2] == '$')
            {
                Array_Manipulation(array, '$');
            }
else              {
j = 2;
                while(production[count][j] != '\0')
                {
                    x = 0;
                    Find_First(temporary_result, production[count][j]);
for(k = 0; temporary_result[k] != '\0'; k++)
                    {
                        Array_Manipulation(array,temporary_result[k]);
                    }
                    for(k = 0; temporary_result[k] != '\0'; k++)
                    {
                        if(temporary_result[k] == '$')
                        {
x = 1;
break;
                        }
}
if(!x)
{
                        break;
                    }
j++;
                }
            }
        }
    }
    return;
}

void Array_Manipulation(char array[], char value)
```

```
{     int
temp;
    for(temp = 0; array[temp] != '\0'; temp++)
    {
        if(array[temp] == value)
        {
            return;
        }
    }
    array[temp] = value;
array[temp + 1] = '\0';
}
```

OUTPUT:



CODE FOR FOLLOW:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h> char
gram[10][10],res[10];
int k=0;
void first(char g,int n)
```

```
{
if(!(isupper(g)))
res[k++]=g; else
{
for(int i=0;i<n;i++)
{
if(gram[i][0]==g)
{
if(islower(gram[i][3]))
res[k++]=gram[i][3]; else
first(gram[i][3],n);
}
}
}
}
void follow(char c,int n)
{
if(gram[0][0]==c)
res[k++]='$'; for(int
i=0;i<n;i++)
{
for(int j=3;j<strlen(gram[i]);j++)
{
if(gram[i][j]==c)
{
if(gram[i][j+1]!='\0')
first(gram[i][j+1],n); if(gram[i][j+1]=='\0'
&& c!=gram[i][0]) follow(gram[i][0],n);
}
}
}
}
int main()
{
char nt; int
n;
```

```
printf("Enter No of Productions ");
scanf("%d",&n); printf("Enter the
Productions");
for(int i=0;i<n;i++)
{
scanf("%s",gram[i]);
}
for(int i=0;i<n;i++)
{
k=0;
follow(gram[i][0],n);
printf("Follow of of %c is {",gram[i][0]); for(int
i=0;i<k;i++)
{
printf(" %c ",res[i]);
}
printf("}"); printf("\n");
}
return 0;
}
```

OUTPUT:

RESULT: The computation of first and follow was achieved through this program

Shreya Shukla
RA1911003010429

Compiler design LAB EXERCISE 6: Construction of Predictive Parser Table

AIM: To write 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.

CODE:

```
#include<stdio.h>
#include<conio.h>
#include<string.h> int
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; printf("enter
the no. of productions\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 pos\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:p++;
}
else
{
while(st[k][j]!=st[a][0])
{
a++;
}
p=0;  while(ft[a][p]!='\0')
{ if(ft[a][p]!='e')
{
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 pos\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][t]=='e')
{
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],fin[s]);
b++;
}
}
else if(!((st[i][t]>64)&&(st[i][t]<91)))
printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[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],fin[s]); b++;
}
}
s++;
} if(st[i][j]=='|')
j++;
}
}
return 0;
}
```

OUTPUT:

main.c

```c
#include<stdio.h>
#include<conio.h>
#include<string.h>
int 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;
printf("enter the no. of productions\n");
scanf("%d",&n);
printf("enter the productions in a grammar\n");
for(i=0;i<n;i++)
scanf("%s",st[i]);
```

input

```
enter the no. of productions
2
enter the productions in a grammar
S->CC
C->eC |d
first pos
FIRS[S]=e
FIRS[C]=e
follow pos
FOLLOW[S]=$
FOLLOW[C]=$

M[S,e]=S->CC
M[C,$]=C->eC


...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

The program was successfully compiled and run.

Exercise 7

Aim- To implement shift reduce parsing in C++.

Algorithm-Shift reduce parser is a type of bottom-up parser. It uses a stack to hold grammar symbols. A parser goes on shifting the input symbols onto the stack until a handle comes on the top of the stack. When a handle occurs on the top of the stack, it implements reduction.

There are the various steps of Shift Reduce Parsing which are

as follows − ● It uses a stack and an input buffer.

- Insert $ at the bottom of the stack and the right end of the input string in Input Buffer.
- Shift: Parser shifts zero or more input symbols onto the stack until the handle is on top of the stack.
- Reduce: Parser reduce or replace the handle on top of the stack to the left side of production, i.e., R.H.S. of production is popped, and L.H.S is pushed.
- Accept: Step 3 and Step 4 will be repeated until it has identified an error or until the stack includes start symbol (S) and input Buffer is empty, i.e., it contains $.
- Error: Signal discovery of a syntax error that has appeared and calls an error recovery routine.

Code-

```
#include<iostream>
#include<string.h>  using
namespace std;  struct
prodn
{
```

```
        char p1[10];  char
        p2[10];
};
int 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';
        cout<<"Enter the Input String:\n";  cin>>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);
                        cout<<"\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);
                                cout<<"\n"<<stack;
                                j=-1;
                        }
                }
                if(strcmp(stack,"E")==0&&i==strlen(input))
                {
                        cout<<"\n\nAccepted";  break;
```

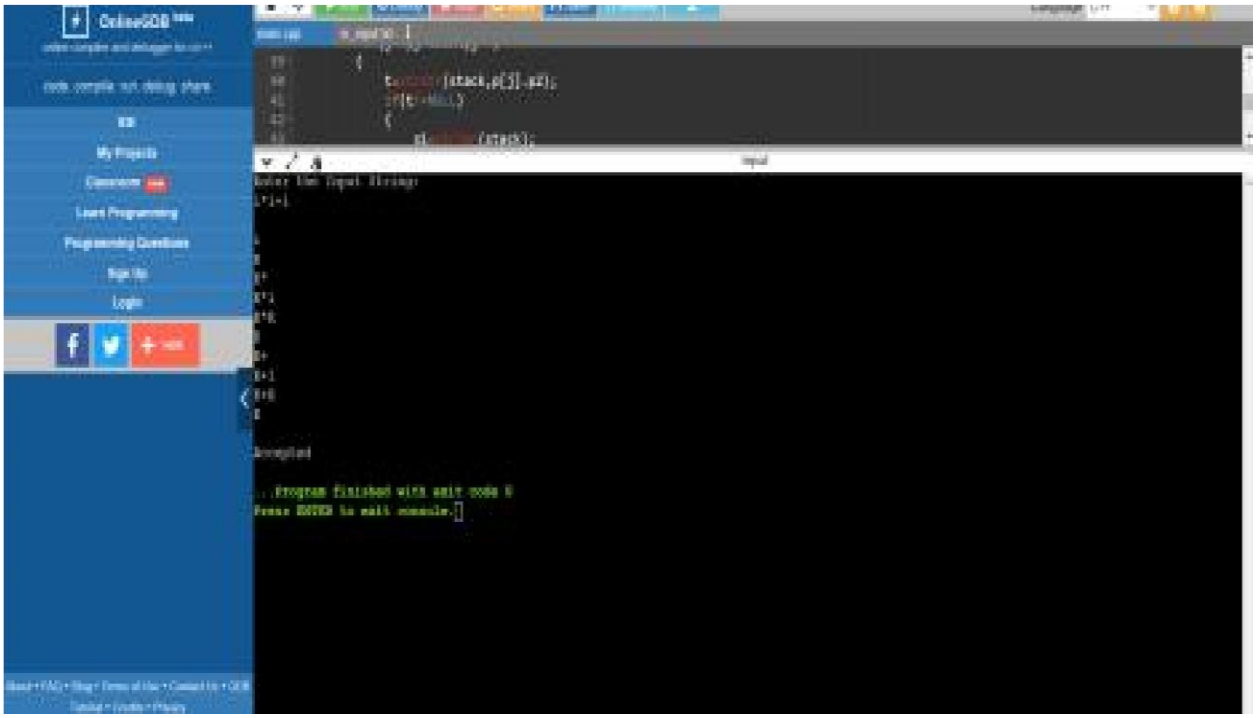
```
                }
                if(i==strlen(input))
                {
                        cout<<"\n\nNot Accepted";
                        break;
                }
        }
        return 0;
}
```

Input production

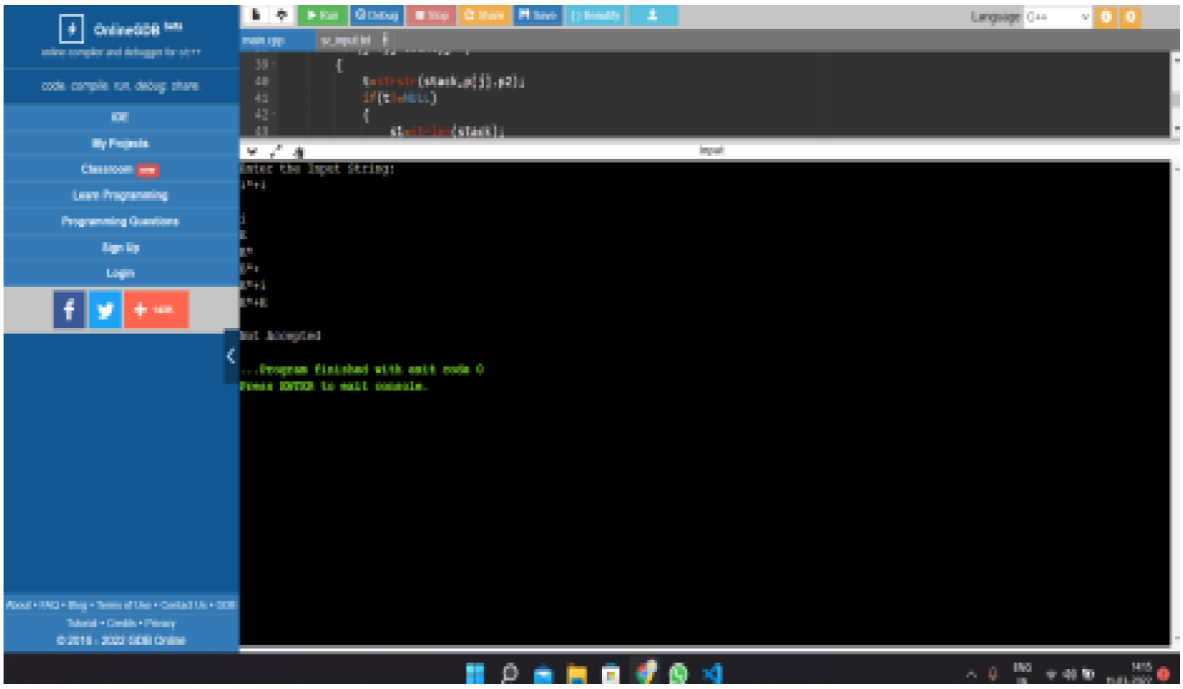
E->E+E

E->E*E E->i

Output



2-



Result-Thus we implemented shift reduce parsing in C++.

Exp 8 - Leading and Trailing

AIM:
To find the leading and trailing of the given grammar

ALGORITHM:

Algorithm to compute LEADING

Input − Context Free Grammar G

Output − LEADING (A) = {a} iff Boolean Array L [A, a] = true

Method − Procedure Install (A, a) will make L (A, a) to true if it was not true earlier.

- begin
- For each non-terminal A and terminal a

  L [A, a] = false ;

- For each production of form A ⟶ aα or A → B a α

  Install (A, a) ;

- While the stack not empty

  Pop top pair (B, a) form Stack ;

  For each production of form A → B α

Install (A, a);

- end

Procedure Install (A, a)

- begin
- If not L [A, a] then

  L [A, a] = true

push (A, a) onto stack.

- end

Algorithm to compute TRAILING

Input − Context Free Grammar G

Output − TRAILING (A) = {a} iff Boolean Array T [A, a] = true

Method

- begin
- For each non-terminal A and terminal a

  T [A, a] = false ;

- For each production of form A ⟶ αa or A → α a B

  Install (A, a) ;

- While the stack not empty

  Pop top pair (B, a) form Stack ;

  For each production of form A → αB

Install (A, a);

- end

Procedure Install (A, a)

- begin
- If not T [A, a] then

  T [A, a] = true push (A, a) onto stack.

- end

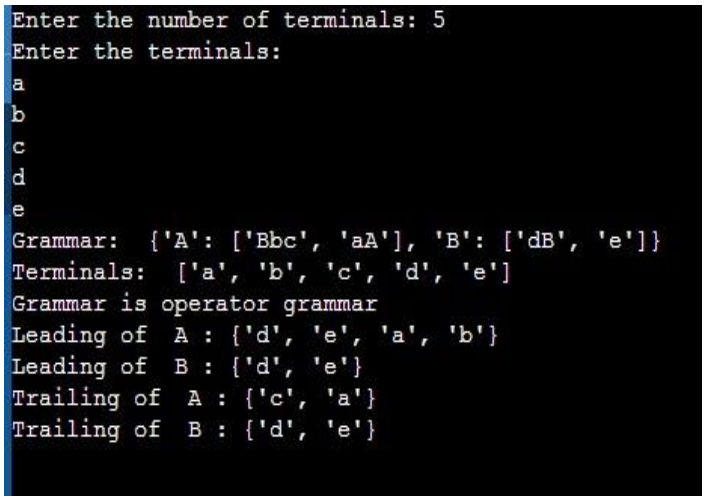
CODE:

```
term_n=int(input("Enter the number of terminals: "))
print("Enter the terminals: ")
term_set=[] for i in
range(term_n):
term_set.append(input())
def
lead(nonterm,gra,arr=[],lea
dset={}):    lead_=set()
for left_v in gra:       if
left_v==nonterm:
```

```
        for prod in gra[left_v]:                if len(prod)==1:
if prod in term_set:                        lead_=lead_|{prod}
else:                    if prod!=nonterm and prod not in arr:
arr.append(prod)                            lead_=lead_ |
lead(prod,gra,arr,leadset)                      elif prod!=nonterm
and prod in leadset:                            lead_=lead_ |
leadset[prod]            else:                  for i in prod:
if i in term_set:                           lead_=lead_|{i}
if prod.index(i)!=0:                            if prod[0]!=nonterm
and prod[0] not in arr:
arr.append(prod[0])
                            lead_=lead_|lead(prod[0],gra,arr,leadset)

                          elif prod[0]!=nonterm and prod[0] in leadset:
lead_=lead_ | leadset[prod[0]]
                      break
   leadset[nonterm]=lead_
return lead_

def trail(nonterm,gra,arr=[],trailset={}):
   trail_=set()    for
left_v in gra:       if
left_v==nonterm:
        for prod in gra[left_v]:                if len(prod)==1:
if prod in term_set:                        trail_=trail_|{prod}
else:                    if (prod!=nonterm) and (prod not
in arr):                        arr.append(prod)
trail_=trail_ | trail(prod,gra,arr,trailset)
elif prod!=nonterm and prod in trailset:
                      trail_=trail_ | trailset[prod]
            else:                for i in prod[::-1]:                    if i in term_set:
trail_=trail_|{i}                       if prod.index(i)!=len(prod)-1:                              if
(prod[len(prod)-1]!=nonterm) and (prod[len(prod)-1] not in arr):
arr.append(prod[len(prod)-1])                               trail_=trail_|trail(prod[len(prod)-
1],gra,arr,trailset)                        elif prod[len(prod)-1]!=nonterm and
prod[len(prod)-1] in trailset:
                            trail_=trail_ | trailset[prod[len(prod)-1]]
break    trailset[nonterm]=trail_
   return trail_

def read_gra(fname):
f=open(fname,"r")
raw_gra=f.read()
lines=raw_gra.split('\n')
gra=dict({})    for i in lines:
words=i.split('->')       for i in
```

```
range(len(words)):
words[i]=words[i].strip()
prod=words[1].split('/')
gra[words[0]]=prod     return
gra

def checkgrammar(gra):
isoperatorgra=True     for
left_v in gra:         for prod
in gra[left_v]:
          j=1            if len(prod)>1:                for i in range(len(prod)-
1):                  if (prod[i] not in term_set) and (prod[j] not in
term_set):                        isoperatorgra=False
                    break
j+=1

          elif prod=='@':
isoperatorgra=False

          if isoperatorgra==False:
             break
       if isoperatorgra==False:
          break     return
isoperatorgra

def printLead(gra):     for i in gra:
print("Leading of ",i,":",lead(i,gra))

def printTrail(gra):     for i in gra:
print("Trailing of ",i,":",trail(i,gra))

gra=read_gra('grammar.txt')
print("Grammar: ",gra) print("Terminals:
",term_set) if(checkgrammar(gra)):
print("Grammar is operator grammar")
   printLead(gra)     printTrail(gra) else:
print("Grammar is not a operator grammar")
```

OUTPUT:



RESULT:

The leading and trailing of the grammar was successfully computed.

Experiment 9- SLR Parsing

Shreya Shukla
RA1911003010429

Algorithm:

- For the given input string write a context free grammar
- Check the ambiguity of the grammar
- Add Augment production in the given grammar
- Create Canonical collection of LR (0) items
- Draw a data flow diagram (DFA)
- Construct a SLR (1) parsing table

Code:

```
#include<bits/stdc++.h>

#define error(x) cerr<<#x<<" = "<<x<<'\n'

using namespace std; set<char> ss;

map<char,vector<vector<char>>> mp;

bool dfs(char i, char org, char last,
map<char,vector<vector<char>>> &mp){

bool rtake = false; for(auto r :

mp[i]){ bool take = true;

for(auto s : r){ if(s == i) break;

if(!take) break;

if(!(s>='A'&&s<='Z')&&s!='e')

{ ss.insert(s); break;

} else if(s == 'e'){

if(org == i||i ==
```

```
last) ss.insert(s);
rtake = true; break;
}
else{ take = dfs(s,org,r[r.size()-
1],mp); rtake |= take;
}
}
}
return rtake;
}
map<int,map<char,set<pair<deque<char>,deque<char>>>>> f;
map<int,vector<pair<int,char>>> g; int num = -1;
void dfs2(char c, char way, int last, pair<deque<char>,deque<char>>
curr){ map<char,set<pair<deque<char>,deque<char>>>> mp2;
int rep = -2; if(last
!= -1){ for(auto q :
g[last]){ if(q.second
== way){ rep =
q.first; mp2 =
f[q.first];
}
}
}
mp2[c].insert(curr); int count = 10; while(count-
-){ for(auto q : mp2){ for(auto r : q.second){
if(!r.second.empty()){
if(r.second.front()>='A'&&r.second.front()<='Z')
{ for(auto s : mp[r.second.front()]){
```

```
deque<char> st,emp; for(auto t : s)
st.push_back(t);
mp2[r.second.front()].insert({emp,st});
}
}
}
}
}
}
for(auto q : f){ if(q.second ==
mp2){
g[last].push_back({q.first,way})
; return;
}
}
if(rep == -2){ f[++num]
= mp2;
if(last != -1)
g[last].push_back({num,way});
}
else{ f[rep] =
mp2;
}
int cc = num; for(auto
q : mp2){ for(auto r :
q.second){
if(!r.second.empty()){
r.first.push_back(r.second.front());
```

```
r.second.pop_front();
dfs2(q.first,r.first.back(),cc,r);
}
}
}
}
int main(){
int i,j;
ifstream fin("inputslr.txt");
string num; vector<int> fs;
vector<vector<int>> a;
char start;
bool flag = 0; cout<<"Grammar:
"<<'\n'; while(getline(fin,num)){
if(flag == 0) start = num[0],flag =
1; cout<<num<<'\n'; vector<char>
temp; char s = num[0];
for(i=3;i<num.size();i++){ if(num[i]
== '|'){ mp[s].push_back(temp);
temp.clear();
}
else temp.push_back(num[i]);
}
mp[s].push_back(temp);
}
map<char,set<char>> fmp; for(auto
q : mp){ ss.clear();
```

```
dfs(q.first,q.first,q.first,mp);
for(auto g : ss)
fmp[q.first].insert(g);
}
cout<<'\n';
cout<<"FIRST:
"<<'\n'; for(auto q :
fmp){ string ans = "";
ans += q.first;

ans += " = {"; for(char
r : q.second){ ans +=
r; ans += ',';
}
ans.pop_back(); ans+="}";
cout<<ans<<'\n';
}
map<char,set<char>> gmp;
gmp[start].insert('$'); int count = 10; while(count--){
for(auto q : mp){ for(auto r : q.second){
for(i=0;i<r.size()-1;i++){ if(r[i]>='A'&&r[i]<='Z'){
if(!(r[i+1]>='A'&&r[i+1]<='Z'))
gmp[r[i]].insert(r[i+1]); else { char temp = r[i+1]; int
j = i+1;
while(temp>='A'&&temp<='Z'){
if(*fmp[temp].begin()=='e'){
for(auto g : fmp[temp]){ if(g=='e')
continue; gmp[r[i]].insert(g);
```

```
}
j++; if(j<r.size()){ temp = r[j];
if(!(temp>='A'&&temp<='Z'))
{ gmp[r[i]].insert(temp);
break;
}
}
else{ for(auto g : gmp[q.first])
gmp[r[i]].insert(g); break;
}
}
else{ for(auto g :
fmp[temp]){
gmp[r[i]].insert(g);
}
break;
}
}
}
}
}
if(r[r.size()-1]>='A'&&r[r.size()-1]<='Z'){
for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
}
}
}
}
}
```

```
cout<<'\n';
cout<<"FOLLOW:
"<<'\n'; for(auto q :
gmp){ string ans = "";
ans += q.first; ans += " =
{"; for(char r : q.second){
ans += r; ans += ',';
}
ans.pop_back(); ans+="}";
cout<<ans<<'\n';
}
string temp = ""; temp+='.';
temp+=start; deque<char> emp;
deque<char> st; st.push_back(start);
dfs2('!','k',-1,{emp,st});
cout<<"\nProductions: "<<'\n'; int cc =
1; set<char> action,go;
map<pair<char,deque<char>>,int>
pos; for(auto q : mp){ go.insert(q.first);
for(auto r : q.second){
cout<<"r"<<cc<<": "; string ans = "";
ans += q.first; ans+="->"; deque<char>
temp; for(auto s : r) ans +=
s,temp.push_back(s);
pos[{q.first,temp}] = cc; for(auto s : r){
if(s>='A'&&s<='Z') go.insert(s); else
action.insert(s);
}
```

```
cout<<ans<<'\n'; cc++;
}
}
cout<<"\nGraph: "<<'\n';
for(auto mp2 : f){
cout<<'\n'; cout<<"l";
cout<<mp2.first<<": \n";
for(auto q :
mp2.second){ string ans
= ""; ans += q.first; ans +=
"->";
for(auto r : q.second){
for(auto t : r.first) ans+=t;
ans+='.'; for(auto t :
r.second) ans+=t; ans+='|';
}
ans.pop_back(); for(auto tt
: ans){ if(tt == '!')
cout<<start<<'\''; else
cout<<tt;
}
cout<<'\n';
}
}
cout<<'\n'; cout<<"Edges: "<<'\n'; for(auto q : g){ for(auto r :
q.second){ cout<<"l"<<q.first<<" -> "<<r.second<<" ->
"<<"l"<<r.first<<"\n";
```

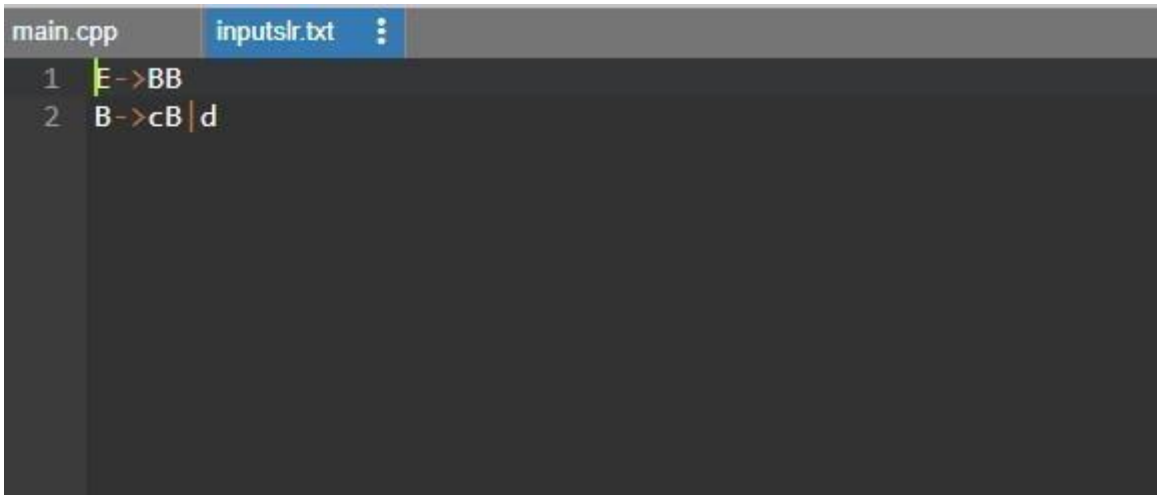
```
}
}
action.insert('$');  cout<<"\nParsing
Table:"<<'\n';   cout<<"St.\t\tAction
& Goto"<<'\n';
int tot = f.size(); cout<<" \t";
for(auto q : action)
cout<<q<<'\t'; for(auto q : go)
cout<<q<<'\t'; cout<<'\n';
for(i=0;i<tot;i++){
cout<<"I"<<i<<'\t'; for(auto q :
action){ if(g.count(i)){ int flag = 0;
for(auto r : g[i]){ if(r.second ==
q){ flag = 1;
cout<<"S"<<r.first<<"\t"; break;
}
}
if(!flag) cout<<"-"<<'\t';
}
else{ int flag =
0; for(auto r :
f[i]){ if(r.first ==
'!'){ if(q == '$'){
cout<<"AC\t";
flag = 1;
}
else cout<<"-\t";
}
```

```
}
if(!flag){ for(auto r : f[i]){ char ccc = r.first;
deque<char> chk = (*r.second.begin()).first;
int cou = 1; for(auto r : gmp[ccc]){ if(q == r){
cout<<"r"<<pos[{ccc,chk}]<<"\t";
}
cou++;
}
}
}
}
}
for(auto q : go){
if(g.count(i)){ int
flag = 0; for(auto r
: g[i]){ if(r.second
== q){ flag = 1;
cout<<r.first<<"\t"
; break;
}
}
if(!flag) cout<<"-"<<'\t';
}
else{ cout<<"-"<<'\t';
}
}
cout<<'\n';
}
```

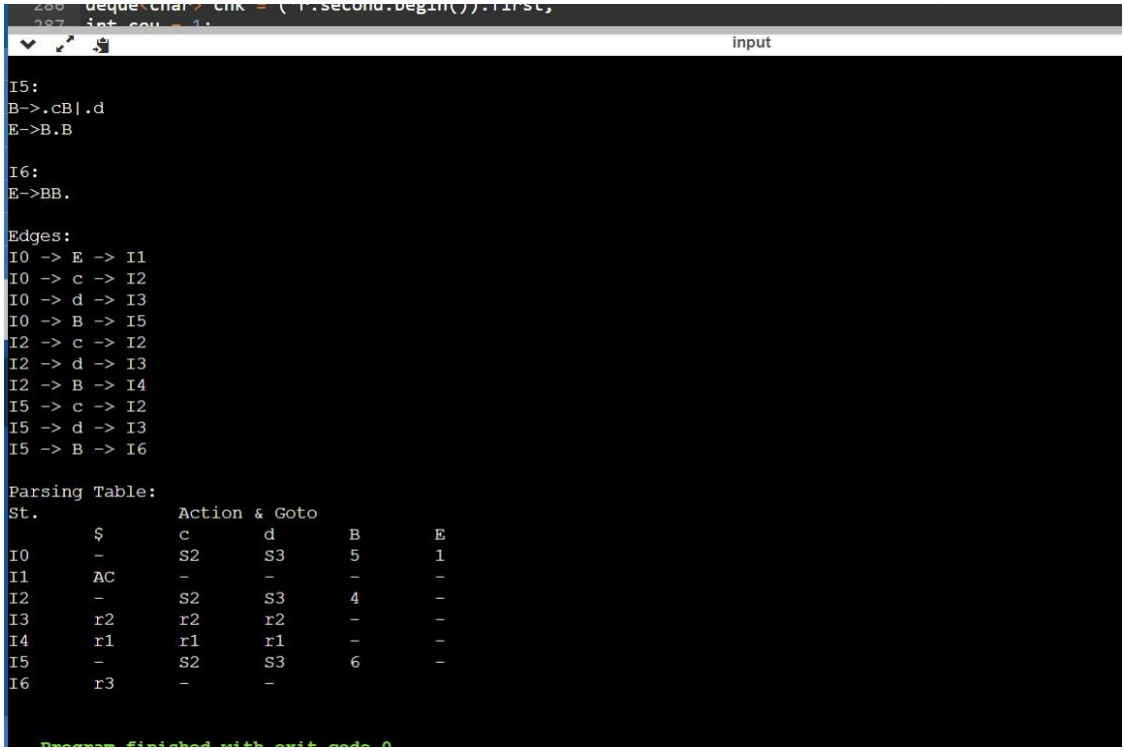
return 0;

}

Input:



Output:



Result:

Thus we implemented SLR parsing table in C++.

EXPERIMENT 10

INTERMEDIATE CODE GENERATION (PREFIX, POSTFIX)

SHREYA SHUKLA

RA1911003010429

AIM: To convert a given input infix expression to prefix and postfix expression.

CODE:

```
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:

RESULT: The given expression was successfully converted into postfix and prefix expression using python language.

EXPERIMENT 11

INTERMEDIATE CODE GENERATION-QUADRUPLE, TRIPLE, INDIRECT TRIPLE

RA1911003010429

SHREYA SHUKLA

Aim:

A program to implement 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:

- Invoke a function getreg to find out the location L where the result of computation b op c should be stored.
- 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.
- 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.
- 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:

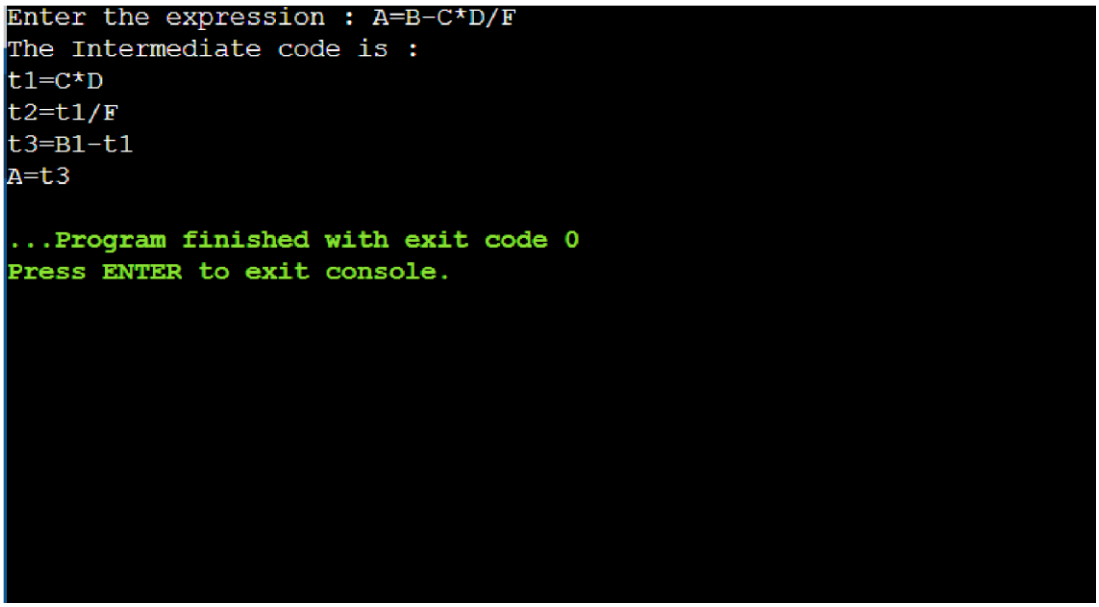
```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h
> void small();
void dove(int i);
int
```

```
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("The 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("t%d=%s*%s\n",c,a,b);
   if(j[i]=='/')
```

```
        printf("t%d=%s/%s\n",c,a,b);
    if(j[i]=='+')
        printf("t%d=%s+%s\n",c,a,b);if(j[i]=='-')
    printf("t%d=%s-%s\n",c,a,b);
    if(j[i]=='=')
        printf("%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);
}
```

Input: A=B-C*D/F

Output:



Result: A program to implement intermediate code generation - Quadruple, Triple, Indirect triple has been compiled and run successfully.

EXPERIMENT 12

A Simple Code Generator, Implementation of DAG, Implementation of Global Data Flow Analysis, Implement storage allocation Strategies

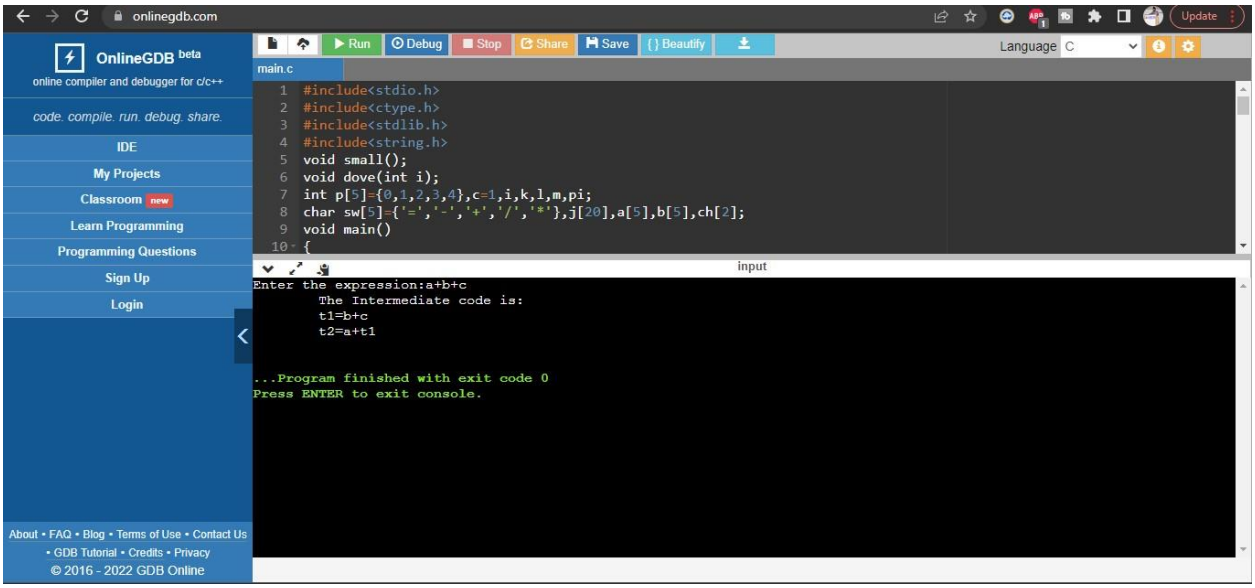
SHREYA SHUKLA
RA1911003010429

1) INTERMEDIATE CODE GENERATION

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#include<graphics.h>
typedef struct { char
var[10]; int alive;
}
regist; regist preg[10]; void
substring(char exp[],int st,int end)
```

```
{ int i,j=0;
char dup[10]="";
for(i=st;i<end;i++)
dup[j++]=exp[i]; dup[j]='0';
strcpy(exp,dup);
} int getregister(char var[])
{ int i;
for(i=0;i<10;i++)
{
if(preg[i].alive==0)
{
strcpy(preg[i].var,var); break;
}}
return(i);
}
void getvar(char exp[],char v[])
{ int i,j=0;
char var[10]="";
for(i=0;exp[i]!='\0';i++)
if(isalpha(exp[i]))
var[j++]=exp[i]; else
break; strcpy(v,var);
}
void main()
{ char basic[10][10],var[10][10],fstr[10],op;
int i,j,k,reg,vc,flag=0; clrscr();
printf("\nEnter the Three Address Code:\n"); for(i=0;;i++)
{
gets(basic[i]);
if(strcmp(basic[i],"exit")==0) break;
}
printf("\nThe Equivalent Assembly Code is:\n"); for(j=0;j<i;j++)
{
getvar(basic[j],var[vc++]); strcpy(fstr,var[vc-1]);
substring(basic[j],strlen(var[vc-1])+1,strlen(basic[j]));
getvar(basic[j],var[vc++]); reg=getregister(var[vc-1]);
if(preg[reg].alive==0)
{
```

```
printf("\nMov R%d,%s",reg,var[vc-1]);
preg[reg].alive=1;
}
op=basic[j][strlen(var[vc-1])]; substring(basic[j],strlen(var[vc-1])+1,strlen(basic[j]));
getvar(basic[j],var[vc++]); switch(op)
{ case '+': printf("\nAdd"); break;
case '-': printf("\nSub"); break; case
'*': printf("\nMul"); break; case '/':
printf("\nDiv"); break;
}
flag=1;
for(k=0;k<=reg;k++)
{ if(strcmp(preg[k].var,var[vc-1])==0)
{
printf("R%d, R%d",k,reg);
preg[k].alive=0; flag=0;
break;
}} if(flag)
{
printf(" %s,R%d",var[vc-1],reg);
printf("\nMov %s,R%d",fstr,reg);
}strcpy(preg[reg].var,var[vc-3]);
getch();


}}
```



2) IMPLEMENTATION OF DAG

Code :

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<conio.h> void
main()
{
struct da
{
int ptr,left,right; char label; }dag[25];
int ptr,l,j,change,n=0,i=0,state=1,x,y,k;
char store,*input1,input[25],var;
clrscr(); for(i=0;i<25;i++)
{
dag[i].ptr=NULL; dag[i].left=NULL;
dag[i].right=NULL;
dag[i].label=NULL;
}
printf("\n\nENTER THE EXPRESSION\n\n");
scanf("%s",input1);
/*EX:((a*b-c))+((b-c)*d)) like this give with paranthesis.limit
is 25 char ucan change that*/ for(i=0;i<25;i++)
input[i]=NULL; l=strlen(input1); a:
```

```
for(i=0;input1[i]!=')';i++);
for(j=i;input1[j]!='(';j--);
for(x=j+1;x<i;x++)
if(isalpha(input1[x]))
input[n++]=input1[x]; else
if(input1[x]!='0')
store=input1[x];
input[n++]=store;
for(x=j;x<=i;x++) input1[x]='0';
if(input1[0]!='0')goto a;
for(i=0;i<n;i++)
{
dag[i].label=input[i]; dag[i].ptr=i;
if(!isalpha(input[i])&&!isdigit(input[i]))
{
dag[i].right=i-1;
ptr=i;
var=input[i-1];
if(isalpha(var))
ptr=ptr-2; else
{
ptr=i-1; b:
if(!isalpha(var)&&!isdigit(var))
{
ptr=dag[ptr].left;
var=input[ptr]; goto
b;
}
else ptr=ptr-1;
}
dag[i].left=ptr;
}
}
printf("\n SYNTAX TREE FOR GIVEN EXPRESSION\n\n"); printf("\n\n PTR
\t\t LEFT PTR \t\t RIGHT PTR \t\t LABEL\n\n"); for(i=0;i<n;i++)/* draw the
syntax tree for the following output with pointer value*/
printf("\n%d\t%d\t%d\t%c\n",dag[i].ptr,dag[i].left,dag[i].right,dag[i].label);
getch(); for(i=0;i<n;i++)
{
```

```
for(j=0;j<n;j++)
{
if((dag[i].label==dag[j].label&&dag[i].left==dag[j].left)&&dag[i].right==dag[j].right)
{
for(k=0;k<n;k++)
{
if(dag[k].left==dag[j].ptr)dag[k].left=dag[i].ptr;
if(dag[k].right==dag[j].ptr)dag[k].right=dag[i].ptr;
}
dag[j].ptr=dag[i].ptr;
}
}
}
printf("\n DAG FOR GIVEN EXPRESSION\n\n"); printf("\n\n PTR \t LEFT PTR \t
RIGHT PTR \t LABEL \n\n"); for(i=0;i<n;i++)/*draw DAG for the following output
with pointer value*/ printf("\n
%dt\t%d\t\t%d\t\t%c\n",dag[i].ptr,dag[i].left,dag[i].right,dag[i].label); getch();
}
```

OUTPUT:

```
ENTER THE EXPRESSION
((a*b-c))+((b-c)*d))

 SYNTAX TREE FOR GIVEN EXPRESSION
 PTR        LEFT PTR          RIGHT PTR          LABEL

0       0       0       a
1       0       0       b
2       0       0       c
3       1       2       -
4       0       3       -

DAG FOR GIVEN EXPRESSION
 PTR    LEFT PTR      RIGHT PTR     LABEL

 0          0           0           a
 1          0           0           b
 2          0           0           c
 3          1           2           -
 4          0           3           -
```

3) IMPLEMENTATION OF GLOBAL DATA FLOW ANALYSIS

Code :

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

```
#include <conio.h>
#include <string.h > struct
op
{
char l[20]; char
r[20];
}
op[10], pr[10];

void main()
{
int a, i, k, j, n, z = 0, m, q,lineno=1;
char * p, * l; char temp, t; char *
tem;char *match;
clrscr(); printf("enter no of
values"); scanf("%d", & n);
for (i = 0; i < n; i++)
{
printf("\tleft\t");
scanf("%s",op[i].l);
printf("\tright:\t"); scanf("%s",
op[i].r);
}
printf("intermediate Code\n"); for
(i = 0; i < n; i++)
{   printf("Line No=%d\n",lineno);
printf("\t\t\t%s=", op[i].l); printf("%s\n",
op[i].r);lineno++;
}
printf("***Data Flow Analysis for the Above Code ***\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
match=strstr(op[j].r,op[i].l); if(match)
{
printf("\n %s is live at  %s \n ", op[i].l,op[j].r);
}
```

```
}
}}
```

OUTPUT:

```
Enter no of values
4
      left   a
      right:  a+b
      left   b
      right:  a+c
      left   c
      right:  a+b
      left   d
      right:  b+c+d
 Line No=1
 a=a+b
 Line No=2
 b=a+c
Line No=3
                c=a+b
Line No=4
                d=b+c+d
***Data Flow Analysis for the Above Code ***

 a is live at  a+b

 a is live at  a+c

 a is live at  a+b

 b is live at  a+b

 b is live at  a+b

 b is live at  b+c+d

 c is live at  a+c

 c is live at  b+c+d

 d is live at  b+c+d
```

4) IMPLEMENTATION OF STORAGE ALLOCATION STRATEGIES

Code :

```
#include<stdio.h>
#include<conio.h> #include<stdlib.h>
```

```
#define    TRUE    1
#define    FALSE    0
typedef struct Heap
{ int data; struct
Heap *next;
}node; node
*create(); void
main()
{ /*local declarations*/
int choice,val; char ans;
node *head; void
display(node *); node
*search(node *,int); node
*insert(node *); void
dele(node **);
head=NULL; do
{ clrscr();
printf("\n Program to perform various operations on heapusing dynamic memory
management");
printf("\n1.Create");
printf("\n2.Display"); printf("\n3.Insert
an element in a list"); printf("\n4.Delete
an element from list");
printf("\n5.Quit"); printf("\n Enter Your
Choice(1-5)"); scanf("%d",&choice);
switch(choice) { case 1:head=create();
break; case 2:display(head);
break; case 3:head=insert(head);
break; case 4:dele(&head);
break; case 5:exit(0);
default:clrscr(); printf("Invalid
Choice,Try again"); getch();
}}
while(choice!=5);
}
node *create()
{
node *temp,*new1,*head;
```

```
int val,flag; char
ans='y'; node
*get_node();
temp=NULL;
flag=TRUE;
/*flag to indicate whether a new node is created for the first time or not*/ do
{
printf("\n Enter the Element"); scanf("%d",&val);
/*allocate new node*/ new1=get_node();
if(new1==NULL) printf("\n Memory is not
allocated"); new1-> data=val; if (flag==TRUE)/*
Executed only for the first time*/
{
head=new1; temp=head; /*head is the first node
in the heap*/ flag=FALSE;
} else
{ /*temp keeps track of the most recently created node*/ temp->next=new1;
temp=new1;
}
printf("\nDo you want to enter more elements?(y/n)");
ans=getch(); }while(ans=='y'); printf("\nThe list is
created"); getch(); clrscr(); return head;
}
node *get_node()
{
node *temp;
temp=(node*)malloc(sizeof(node));
//using the mem. Allocation function
temp->next=NULL; return temp;
}
void display(node*head)
{
node *temp; temp=head;
if(temp==NULL)
{
printf("\n The list is empty\n");
getch(); clrscr(); return;
}
```

```
while(temp!= NULL)
{
printf("%d->",temp-> data);
temp=temp->next;
}
printf("NULL");
getch(); clrscr();
}
node *search(node *head,int key)
{
node *temp; int
found;
temp=head; if
(temp==NULL)
{
printf("The linked list is empty\n");
getch(); clrscr();
return NULL;
}
found=FALSE;
while((temp!=NULL)&&(found==FALSE))
{ if(temp->data != key) temp = temp-
>next; else found = TRUE;
} if(found == TRUE)
{
printf("\n The Elements is present in the list\n");
getch(); return temp;
} else printf("\n The Element is not present in the
list\n"); getch(); return NULL;

}



node *insert(node *head)
{ int choice; node
*insert_head(node*); void
```

```
insert_after(node*); void
insert_last(node*);
printf("\nInsert a node as a
head node");
printf("\nInsert a node as a
last node");
printf("\nInsert a node as
at the intermediate
position in the list ");
printf("\nEnter your choice
for insertion of node ");
scanf("%d",&choice);
switch(choice) { case
1:head =
insert_head(head); break;
case 2:insert_last(head);
break; case 3:insert_after
(head); break;
}
return head;
} /*Insertion of node at first position*/
node *insert_head(node*head)
{
node *New,*temp; New = get_node(); printf ("\n Enter
the element which you want to insert ");
scanf("%d",&New->data); if(head == NULL) head = New;
else
{ temp=head; New->next
= temp; head= New;
}
return head;
} /*Insertion of node at last position*/
void insert_last(node *head)
{
node *New,*temp; New = get_node(); printf ("\n Enter
the element which you want to insert ");
scanf("%d",&New->data); if(head == NULL)
{
head = New;
```

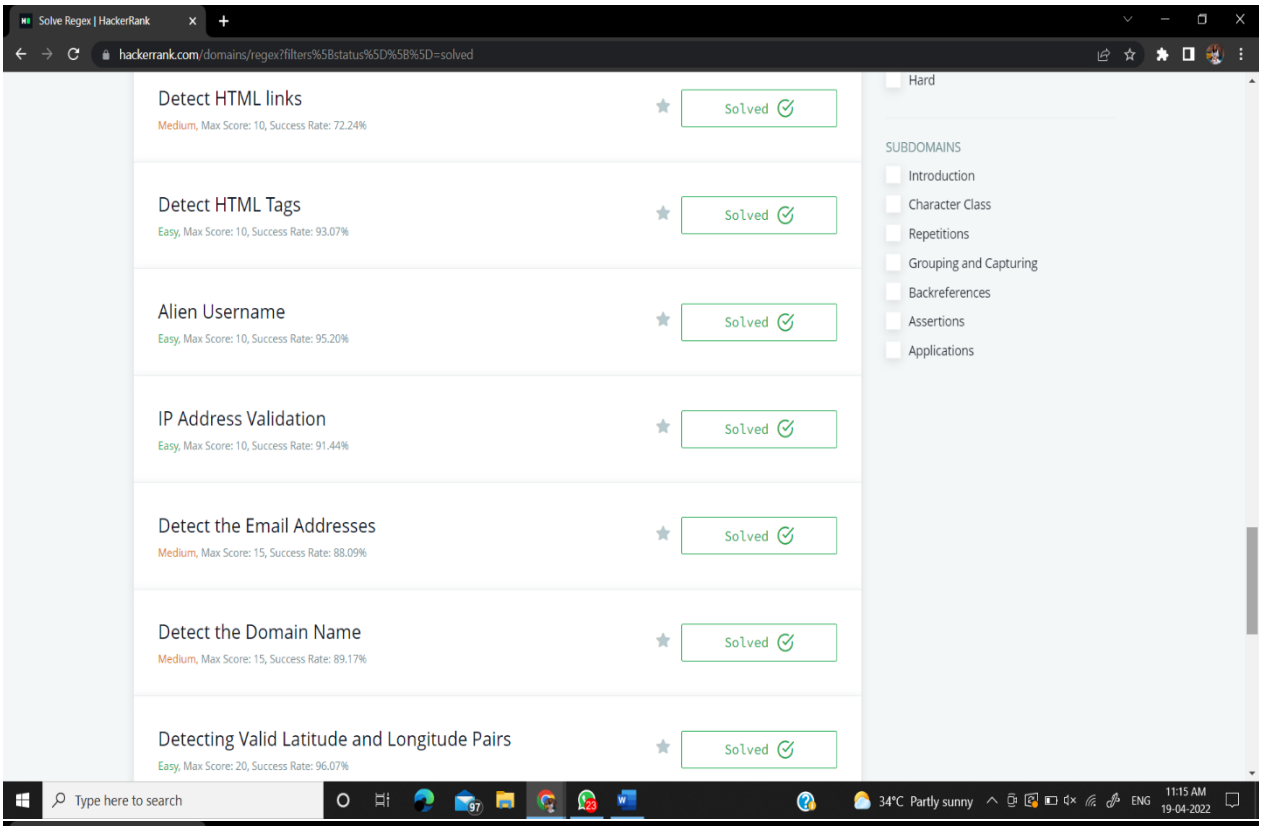
```
} else
{ temp=head; while(temp-
>next!=NULL) temp=temp->next;
temp->next=New;


New->next=NULL;


}}
/*Insertion of node at intermediate position*/ void
insert_after(node *head)
{ int key;
node *New,*temp; New = get_node(); printf("Enter the
element after which you want to insert ");
scanf("%d",&key); temp=head; do
{ if(temp->data==key)
{
printf ("Enter element which you want to insert ");
scanf("%d",&New->data); New->next=temp-
>next; temp->next=New; return;
} else
temp=temp->next;
}while(temp!=NULL);
}
node *get_prev(node *head,int val)
{
node *temp,*prev;
int flag;
temp = head; if(temp
== NULL) return
NULL; flag = FALSE;
prev = NULL;
while(temp!=NULL
&& !flag)
{ if(temp->data!=val)
{
prev = temp; temp
= temp->next;
```

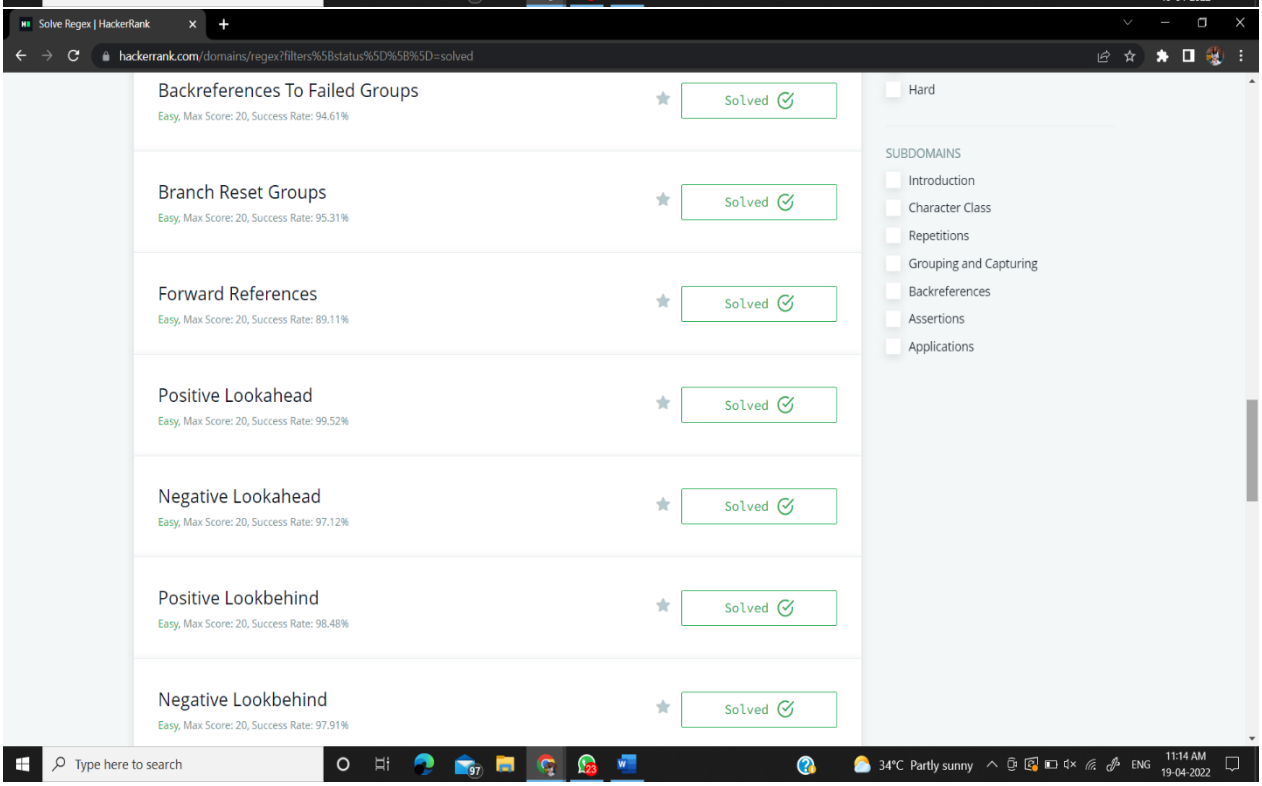
```
} else flag =
TRUE;
} if(flag) /*if Flag is true*/
return prev; else return
NULL;
}
void dele(node **head)
{ int key;
node *New,*temp, *prev;
temp=*head; if (temp==
NULL)
{
printf ("\n The list is empty\n ");
getch(); clrscr(); return; }
clrscr();
printf("\nENTER the Element you want to delete:");
scanf("%d",&key); temp= search(*head,key);
if(temp !=NULL)
{
prev = get_prev(*head,key); if(prev
!= NULL)
{
prev ->next = temp-> next; free(temp);
} else
{
*head = temp->next; free(temp); // using the
mem. Dellocation function
}
printf("\nThe Element is deleted\n");
getch(); clrscr();
}}
```

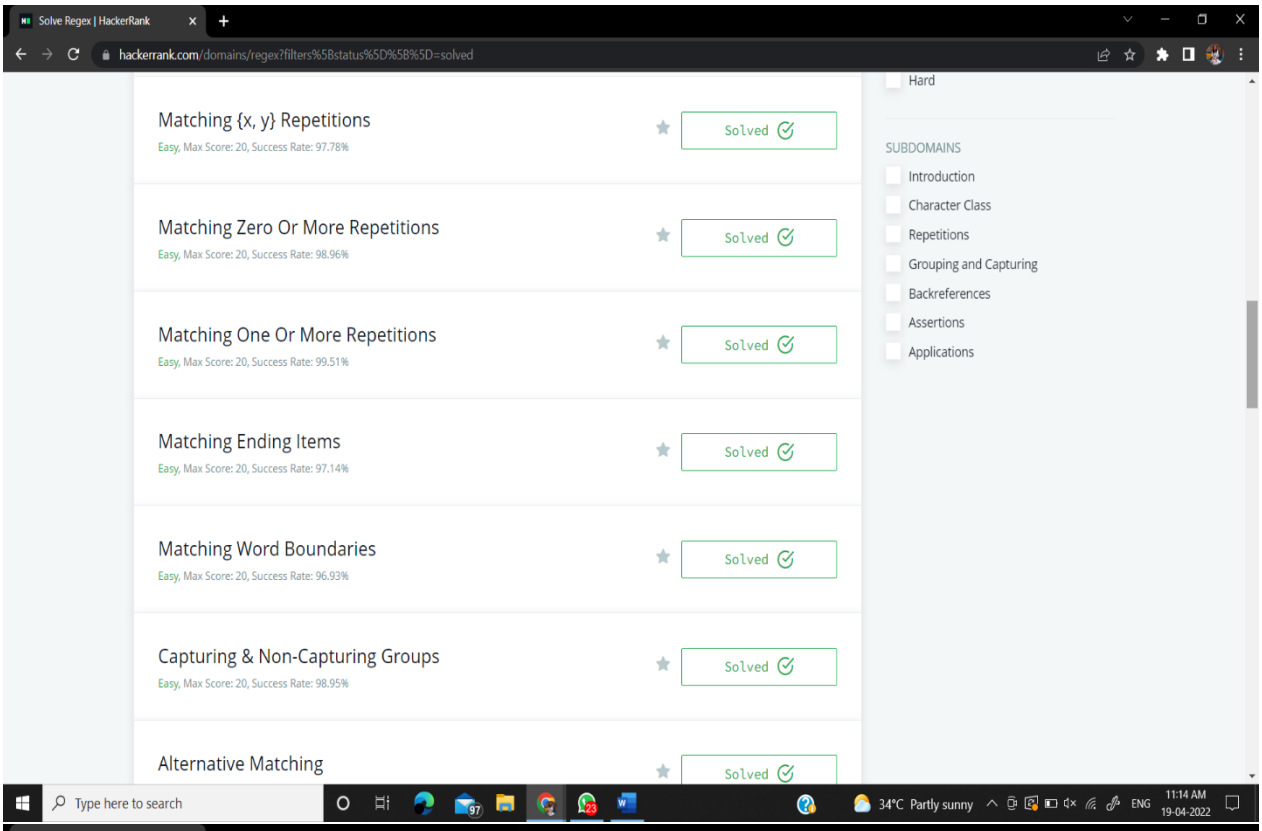
OUTPUT:

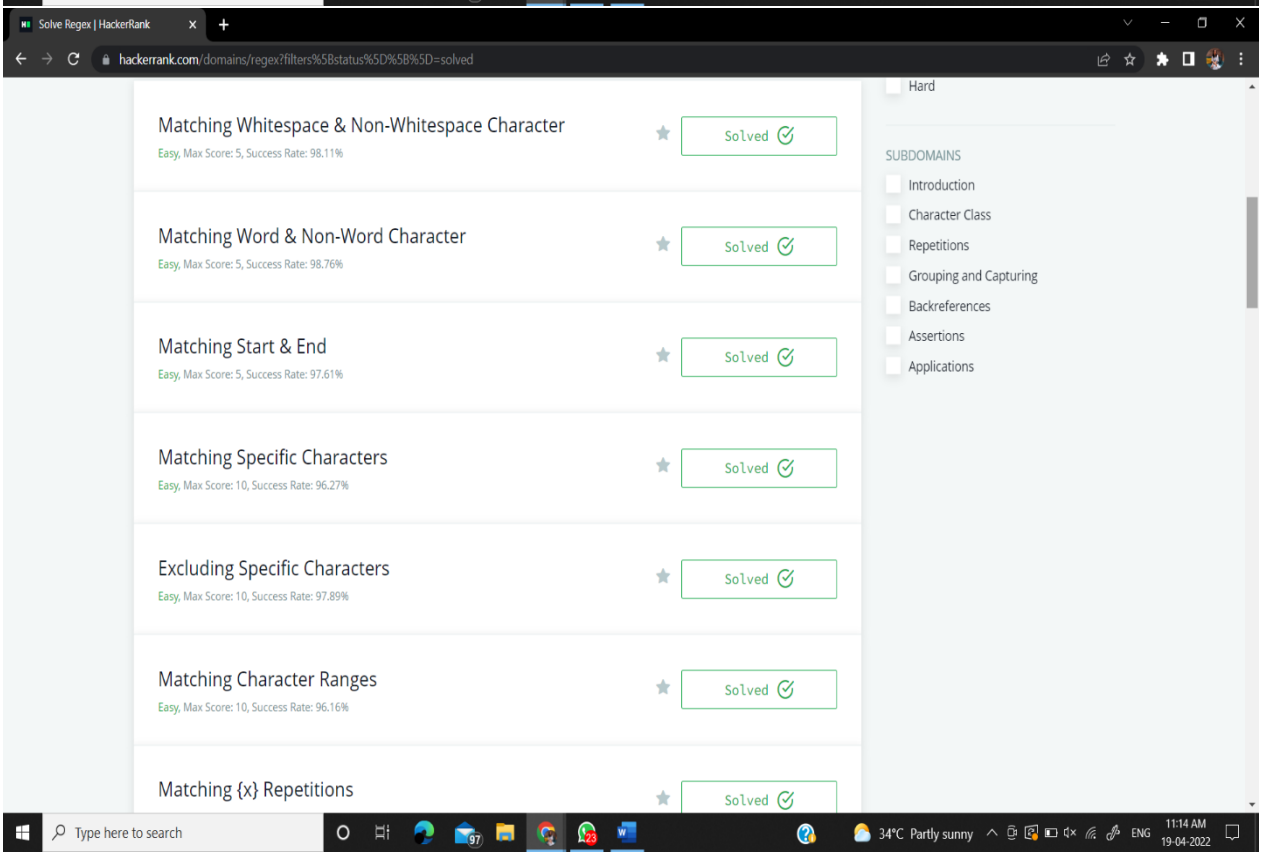
```
Program to perform various operations on heap using Dynamic memory
management.
1.Create
2.Display
3.Insert an element in a list
4. Delete an element from list
5. Quit
Enter your choice(1-5) 1
Enter the element: 10
Do you want to enter more elements? (y/n) y
Enter the element:20
Do you want to enter more elements?(y/n)y
Enter the element:30
Do you want to enter more elements?(y/n)n
The List is created
Program to perform various operations on Heap using Dynamic memory
management.
1. Create
2. Display
3. Insert an element in a list
4. Delete an element from list
5. Quit
Enter your choice(1-5) 4
Enter the element you want to delete: 20
The element is present in the list
The element is deleted
Program to perform various operations on Heap using Dynamic memory
management.
1. Create
2. Display
3. Insert an element in a list
4. Delete an element from list
5. Quit
Enter your choice(1-5) 2
10-> 30-> NULL
```

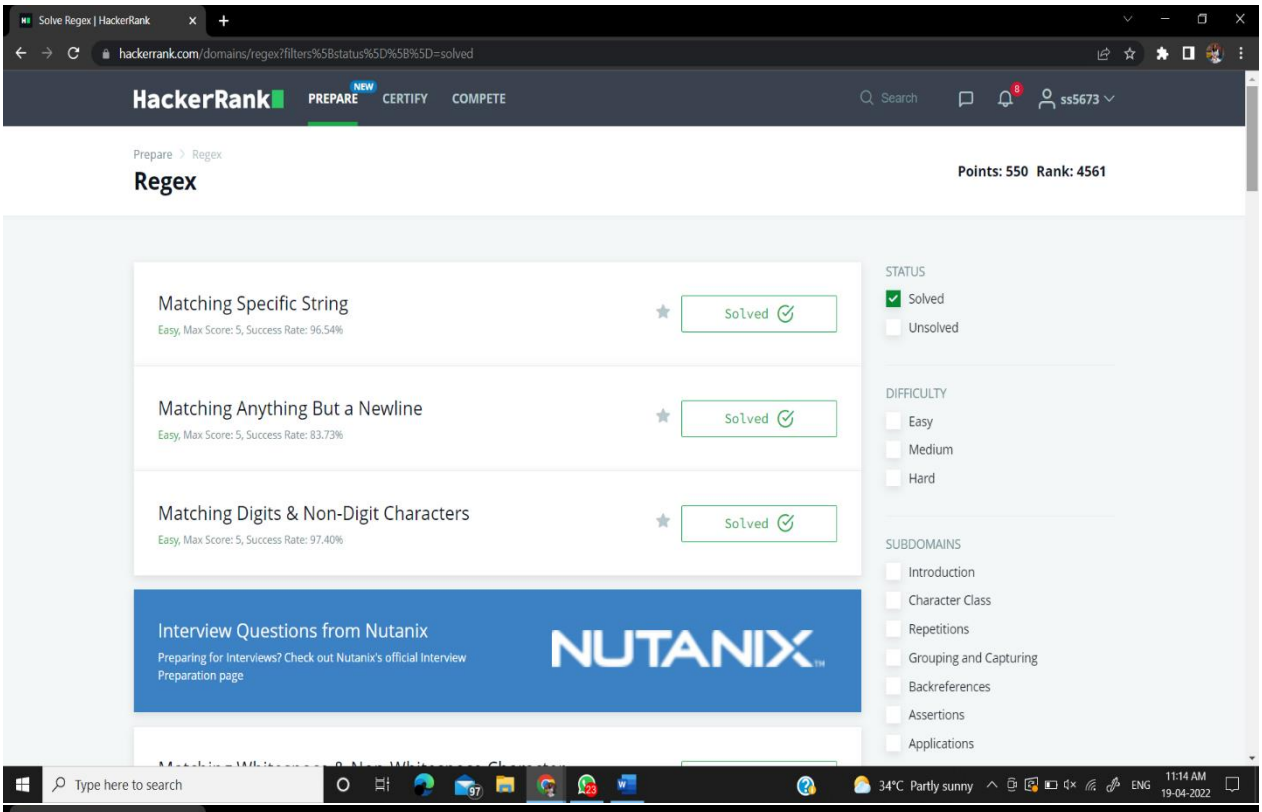
HACKERRANK REGEX

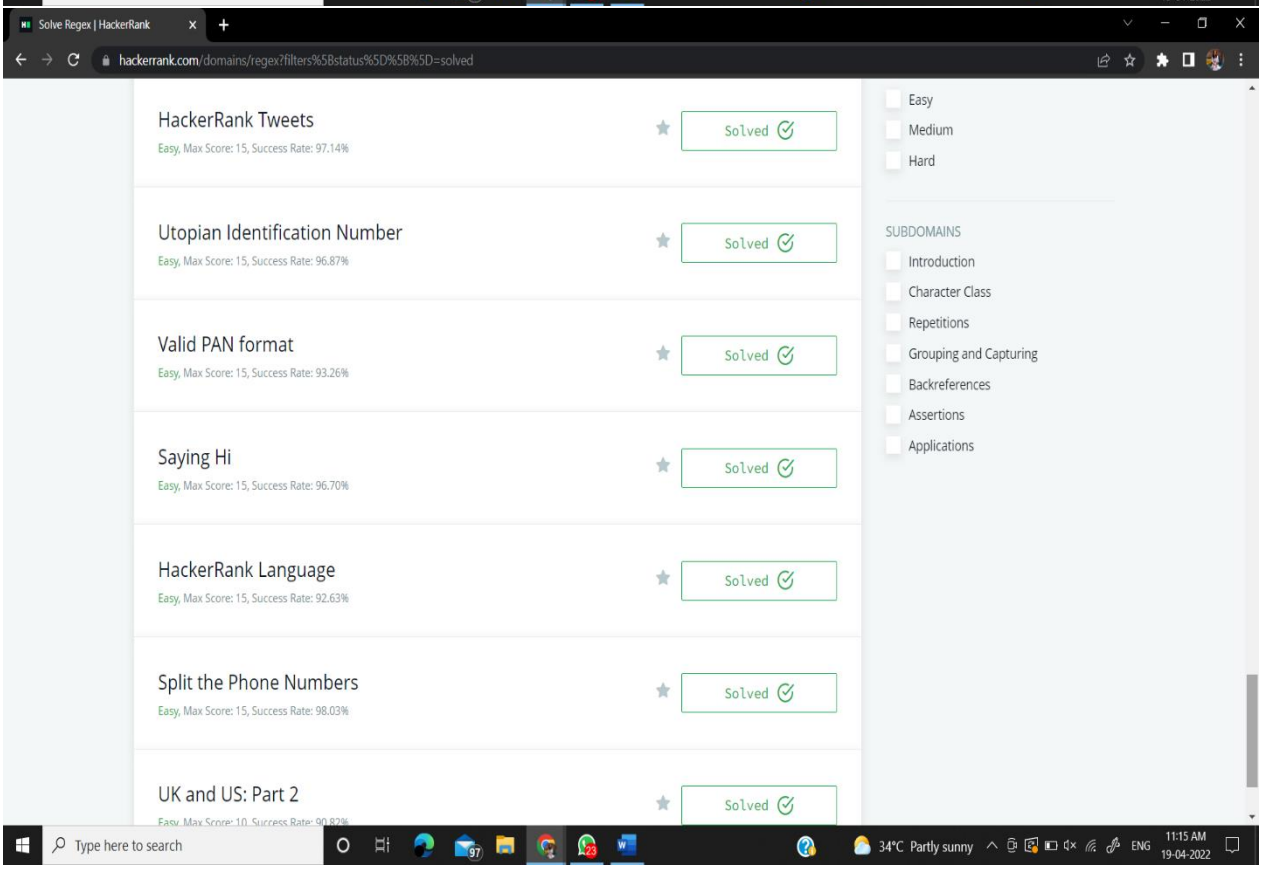





Solve Regex | HackerRank
hackerrank.com/domains/regex?filters%5Bstatus%5D%5B%5D=solved
Hard
SUBDOMAINS
Introduction
Character Class
Repetitions
Grouping and Capturing
Backreferences
Assertions
Applications
Matching {x, y} Repetitions
Easy, Max Score: 20, Success Rate: 97.78%
Solved
Matching Zero Or More Repetitions
Easy, Max Score: 20, Success Rate: 98.96%
Solved
Matching One Or More Repetitions
Easy, Max Score: 20, Success Rate: 99.51%
Solved
Matching Ending Items
Easy, Max Score: 20, Success Rate: 97.14%
Solved
Matching Word Boundaries
Easy, Max Score: 20, Success Rate: 96.93%
Solved
Capturing & Non-Capturing Groups
Easy, Max Score: 20, Success Rate: 98.95%
Solved
Alternative Matching
Solved
Type here to search
34°C Partly sunny
ENG
11:14 AM
19-04-2022


Solve Regex | HackerRank
hackerrank.com/domains/regex?filters%5Bstatus%5D%5B%5D=solved
Hard
SUBDOMAINS
Introduction
Character Class
Repetitions
Grouping and Capturing
Backreferences
Assertions
Applications
Matching Whitespace & Non-Whitespace Character
Easy, Max Score: 5, Success Rate: 98.11%
Solved
Matching Word & Non-Word Character
Easy, Max Score: 5, Success Rate: 98.76%
Solved
Matching Start & End
Easy, Max Score: 5, Success Rate: 97.61%
Solved
Matching Specific Characters
Easy, Max Score: 10, Success Rate: 96.27%
Solved
Excluding Specific Characters
Easy, Max Score: 10, Success Rate: 97.89%
Solved
Matching Character Ranges
Easy, Max Score: 10, Success Rate: 96.16%
Solved
Matching {x} Repetitions
Solved
Type here to search
34°C Partly sunny
ENG
11:14 AM
19-04-2022


Solve Regex | HackerRank
hackerrank.com/domains/regex?filters%5Bstatus%5D%5B%5D=solved
HackerRank
PREPARE
CERTIFY
COMPETE
Search
ss5673
Prepare > Regex
Regex
Points: 550 Rank: 4561
Matching Specific String
Easy, Max Score: 5, Success Rate: 96.54%
Solved
Matching Anything But a Newline
Easy, Max Score: 5, Success Rate: 83.73%
Solved
Matching Digits & Non-Digit Characters
Easy, Max Score: 5, Success Rate: 97.40%
Solved
Interview Questions from Nutanix
Preparing for Interviews? Check out Nutanix's official Interview Preparation page
NUTANIX
STATUS
Solved
Unsolved
DIFFICULTY
Easy
Medium
Hard
SUBDOMAINS
Introduction
Character Class
Repetitions
Grouping and Capturing
Backreferences
Assertions
Applications
Type here to search
34°C Partly sunny
ENG
11:14 AM
19-04-2022


Solve Regex | HackerRank
hackerrank.com/domains/regex?filters%5Bstatus%5D%5B%5D=solved
HackerRank Tweets
Easy, Max Score: 15, Success Rate: 97.14%
Solved
Utopian Identification Number
Easy, Max Score: 15, Success Rate: 96.87%
Solved
Valid PAN format
Easy, Max Score: 15, Success Rate: 93.26%
Solved
Saying Hi
Easy, Max Score: 15, Success Rate: 96.70%
Solved
HackerRank Language
Easy, Max Score: 15, Success Rate: 92.63%
Solved
Split the Phone Numbers
Easy, Max Score: 15, Success Rate: 98.03%
Solved
UK and US: Part 2
Solved
Easy
Medium
Hard
SUBDOMAINS
Introduction
Character Class
Repetitions
Grouping and Capturing
Backreferences
Assertions
Applications
Type here to search
34°C Partly sunny
ENG
11:15 AM
19-04-2022

MINI PROJECT:

ABSTRACT

Being a programmer requires all-around knowledge of a lot of subjects. One of the most underrated among these would definitely be compiler design. Knowing what goes behind the scenes when a program is run can help one understand the source of errors and warnings (which is definitely a big help since it helps optimize the time taken to run a program successfully). Today, we're going to be looking at one of the steps performed by a compiler before it executes the program: it's called lexical analysis.

INTRODUCTION

During the compilation process, the first step that is undertaken is called **lexical analysis**. During this process, the program typed by the user is shredded to pieces and every token that is a part of it is extracted and stored separately (tokens are the smallest indivisible parts of a program). These tokens need to be classified into particular types before the compilation process can begin.

There are several types of token which are associated with any language. The naming given by the user for several parts of the program like functions and variables is called identifiers. They are called as such because they "identify" a named storage location in the memory. Then comes keywords: a number of words used by the language for some of its functionality (In C++, these include words like cout, cin, if, else, for, break, continue, and so on). Punctuators are used for the construction of expressions and statements. They are useful only when used in conjunction with identifiers or keywords in a statement. Operators are used for performing actual operations with the data (like arithmetic, logical, and shift operations). Literals are constant data that the programs need to deal with, like numbers or alphabets (or a combination of both).

What does a lexical analyzer do? Separation of a program into its tokens and classification of the tokens is the main responsibility of the lexical analyzer.

CODE FOR LEXICAL ANALYSER:

```
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>


using namespace std;

bool isPunctuator(char ch)                    //check if the given character is a
punctuator or not
```

```
{
   if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
      ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
      ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
      ch == '[' || ch == ']' || ch == '{' || ch == '}' ||
      ch == '&' || ch == '|')
      {
         return true;
      }
   return false;
}

bool validIdentifier(char* str)                                  //check if the given identifier
is valid or not
{
   if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
      str[0] == '3' || str[0] == '4' || str[0] == '5' ||
      str[0] == '6' || str[0] == '7' || str[0] == '8' ||
      str[0] == '9' || isPunctuator(str[0]) == true)
      {
         return false;
      }                                                          //if first character of string is
a digit or a special character, identifier is not valid
   int i,len = strlen(str);
   if (len == 1)
   {
      return true;
   }                                                                    //if length is one,
validation is already completed, hence return true
   else
   {
   for (i = 1 ; i < len ; i++)                                  //identifier cannot contain
special characters
   {
      if (isPunctuator(str[i]) == true)
      {
         return false;
      }
   }
   }
   return true;
}
```

```
bool isOperator(char ch)                                        //check if the given
character is an operator or not
{
   if (ch == '+' || ch == '-' || ch == '*' ||
       ch == '/' || ch == '>' || ch == '<' ||
       ch == '=' || ch == '|' || ch == '&')
   {
     return true;
   }
   return false;
}

bool isKeyword(char *str)                                       //check if the given substring
is a keyword or not
{
   if (!strcmp(str, "if") || !strcmp(str, "else") ||
       !strcmp(str, "while") || !strcmp(str, "do") ||
       !strcmp(str, "break") ||  !strcmp(str, "continue")
       || !strcmp(str, "int") || !strcmp(str, "double")
       || !strcmp(str, "float") || !strcmp(str, "return")
       || !strcmp(str, "char") || !strcmp(str, "case")
       || !strcmp(str, "long") || !strcmp(str, "short")
       || !strcmp(str, "typedef") || !strcmp(str, "switch")
       || !strcmp(str, "unsigned") || !strcmp(str, "void")
       || !strcmp(str, "static") || !strcmp(str, "struct")
       || !strcmp(str, "sizeof") || !strcmp(str,"long")
       || !strcmp(str, "volatile") || !strcmp(str, "typedef")
       || !strcmp(str, "enum") || !strcmp(str, "const")
       || !strcmp(str, "union") || !strcmp(str, "extern")
       || !strcmp(str,"bool"))
       {
          return true;
       }
   else
   {
     return false;
   }
}

bool isNumber(char* str)                                        //check if the given
substring is a number or not
{
   int i, len = strlen(str),numOfDecimal = 0;
   if (len == 0)
```

```
    {
        return false;
    }
    for (i = 0 ; i < len ; i++)
    {
        if (numOfDecimal > 1 && str[i] == '.')
        {
            return false;
        } else if (numOfDecimal <= 1)
        {
            numOfDecimal++;
        }
        if (str[i] != '0' && str[i] != '1' && str[i] != '2'
            && str[i] != '3' && str[i] != '4' && str[i] != '5'
            && str[i] != '6' && str[i] != '7' && str[i] != '8'
            && str[i] != '9' || (str[i] == '-' && i > 0))
            {
                return false;
            }
    }
    return true;
}

char* subString(char* realStr, int l, int r)                    //extract the required
substring from the main string
{
    int i;

    char* str = (char*) malloc(sizeof(char) * (r - l + 2));

    for (i = l; i <= r; i++)
    {
        str[i - l] = realStr[i];
        str[r - l + 1] = '\0';
    }
    return str;
}


void parse(char* str)                                           //parse the expression
{
    int left = 0, right = 0;
    int len = strlen(str);
    while (right <= len && left <= right) {
```

```
      if (isPunctuator(str[right]) == false)                    //if character is a digit or an alphabet
         {
            right++;
         }

      if (isPunctuator(str[right]) == true && left == right)          //if character is a punctuator
         {
         if (isOperator(str[right]) == true)
         {
            std::cout<< str[right] <<" IS AN OPERATOR\n";
         }
         right++;
         left = right;
         } else if (isPunctuator(str[right]) == true && left != right
               || (right == len && left != right))                  //check if parsed substring is a
keyword or identifier or number
         {
         char* sub = subString(str, left, right - 1);   //extract substring

         if (isKeyword(sub) == true)
                  {
                     cout<< sub <<" IS A KEYWORD\n";
                  }
         else if (isNumber(sub) == true)
                  {
                     cout<< sub <<" IS A NUMBER\n";
                  }
         else if (validIdentifier(sub) == true
                && isPunctuator(str[right - 1]) == false)
                {
                   cout<< sub <<" IS A VALID IDENTIFIER\n";
                }
         else if (validIdentifier(sub) == false
                && isPunctuator(str[right - 1]) == false)
                {
                   cout<< sub <<" IS NOT A VALID IDENTIFIER\n";
                }

         left = right;
         }
   }
   return;
}
```

```
int main()
{
   char c[100] = "int m = n + 3p";
   parse(c);
   return 0;
}
```

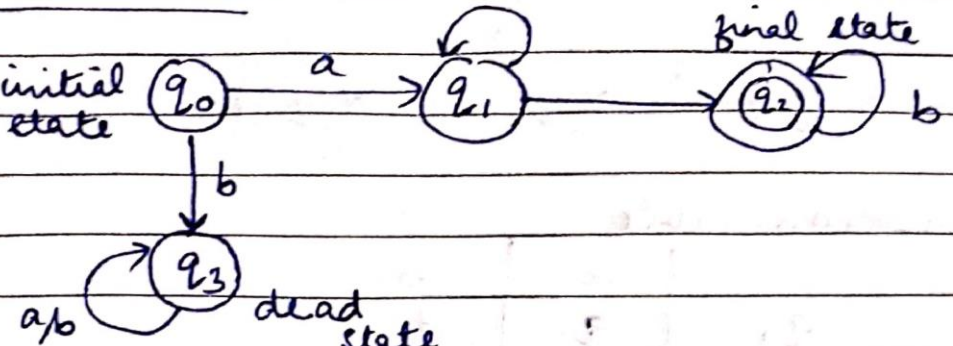
OUTPUT:

ASSIGNMENTS:

Compiler Design

Shreya Shukla
RA1911003010-429

Question 1



	a	b
q_0	q_1	q_3
q_1	q_1	q_2
(q_2)	q_1	q_2
q_3	q_3	q_3

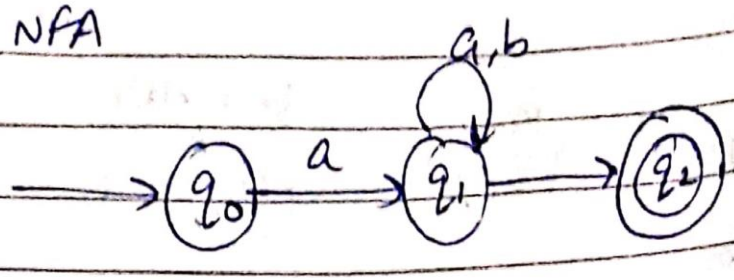
The finite automata model that allows exactly one transition form from a state on an input symbol is called Deterministic Finite Automata (DFA)

For string abbab, it will start from q_0 and go to q_1, For 'a', then for b it will goto q_2, it will remain at q_2. For 'second b' and then return to q_1. For a and then again to q_2, for the final state

Shreya Shukla, RA1911003010429

Question 2

NFA



Transition Table

	a	b
q_0	q_1	ϕ
q_0	q_2	$\{q_1, q_2\}$
q_2	ϕ	ϕ

The finite automata model that allows 0 or 1 or more transitions from a state on an input symbol is called - Not deterministic Automata. (NFA)

For abbab, it will go through,

$$q_0 \rightarrow q_1 \rightarrow q_1 \rightarrow q_1 \rightarrow q_2$$

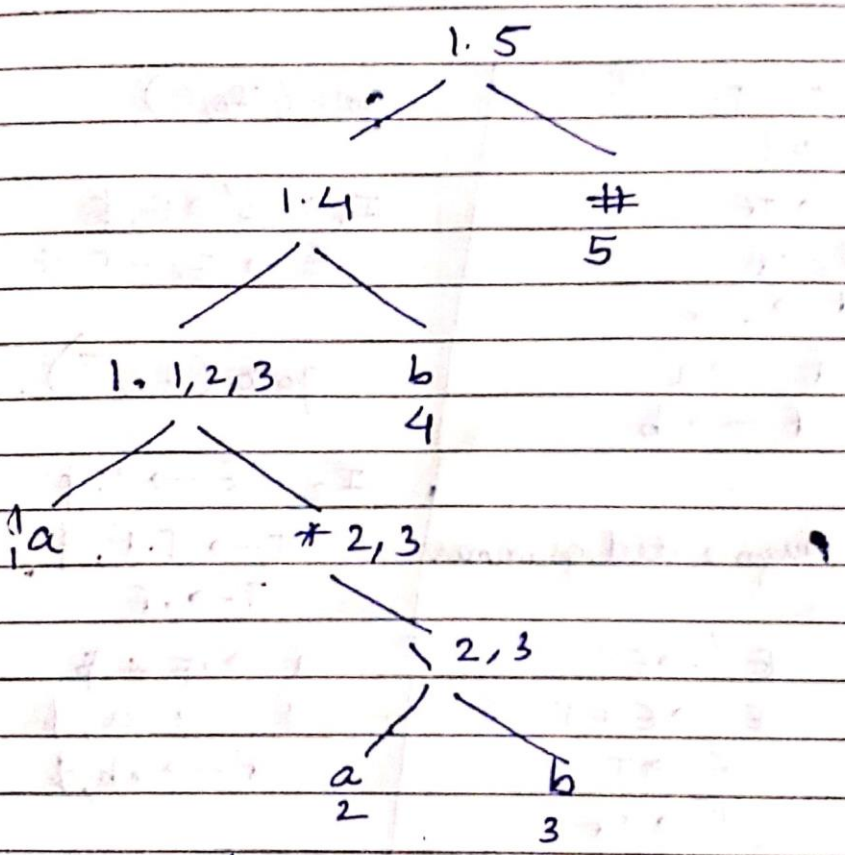
Shreya Shukla, RA1911003010429

Question 3.

$R.E = a(a/b)^* b$

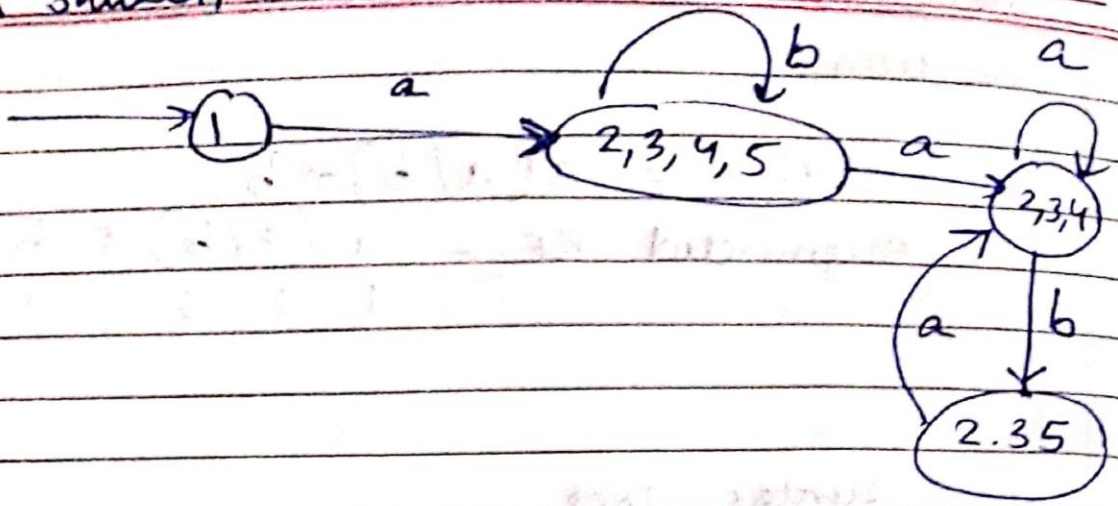
Augmented RE = $a_1 \cdot (a/b)^*_{2\,3}\; b_4 \cdot \#_5$

Syntax Tree:



Node		Follow Pop
a	1	2, 3, 4
a	2	2, 3, 4
b	3	2, 3, 4
b	4	5
#	5	—

Shreya Shukla, RA1911003010429



Question 4

$E \to E+T$	goto (I_0, E)
$E \to T$	
$T \to TF$	I_1 : $E' \to E.\$$
$T \to F$	$E \to E.+T, +$
$F \to F*$	
$F \to a$	goto (I_0, T).
$F \to b$	
	I_2 : $E \to T.\$$
Augmented grammar	$T \to T.F, \$$.
	$T \to .F$
$E' \to E$	$F \to .F* \$$
$E \to E+T$	$F \to .a, \$$
$E \to T$	$F \to .b, \$$
$T \to TF$	
$T \to F$	goto (I_0, F)
$F \to F*$	
$F \to a$	I_3: $T \to F.\$$
$F \to b$	$F \to F.*, \$$

Shreya Shukla, RA1911003010429

Date
Page

LR(1) items

I_0: $E' \to .E, \$$
$E \to .E+T, \$$
$E \to .T, \$$
$T \to .F, \$$
$F \to F*, a/b$
$F \to a, a/b$
$F \to b, a/b.$

goto(I_0, a)

I_4 $F \to a., a/b$

goto(I_0, b)

I_5 : $F \to b., a/b$

goto(I_1, +)

I_6 : $E \to E+.T$
$T \to .TF$
$T \to .F$
$F \to .F*$
$F \to .a$
$T \to .b$

goto(I_2, F)

I_7 : $E \to TF., \$$
$F \to F.*, \$$

goto(I_3, *)

I_8 : $F \to F*.$

State	Action					Goto		
	a	b	*	+	$	E	T	F
0	S_4	S_5				1	2	3
1				S_6	acc			
2	S_4	S_5					7	
3			S_8					
4	r_6	r_6						
5	r_7	r_7						
6	S_4	S_5						
7				S_6				
8					r_5			

Shreya Shukla, RA1911003010429

Question 5:

$S \to aSA$

$S \to \varepsilon$

$A \to bS$

$A \to c$

Augmented grammar

$S' \to S$

$S \to aSA$

$S \to \varepsilon$

$A \to bS$

$A \to c$

$I_0 : S' \to .S | \$$

$S \to .aSA | \$$

$S \to .\varepsilon | \$$

$A \to .bS | \$$

$A \to .c | \$$

goto (I_0, S)

$I_2 : S \to a.SA, \$$

$S \to .\varepsilon, b|c$

goto (I_0, ε)

$I_3 : S \to \varepsilon., \$$

goto (I_0, b)

$I_4, A \to b.S, \$$

goto (I_0, c)

$I_5 : A \to c., b|c$

goto (I_2, S)

$I_6 : S \to aSA, \$$

$A \to .bS, \$$

$A \to .c, \$$

goto (I_4, S)

$I_7 : A \to bS., \$$

goto (I_6, A)

State	Action					Goto	
	a	b	c	ε	$	S	A
0	S_2		S_4	S_5	S_3	1	
1					I_3 acc		
2						6	
3				r_2			
4						7	
5		r_1		r_4			
6		S_4		S_5			
7					r_3		8
8					r_1		