Date: 10th Jan 2022

Exercise 1

Implementation of Lexical Analyser

Aim:

To write a program and execute the Lexical Analyzer and implement the same.

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 :

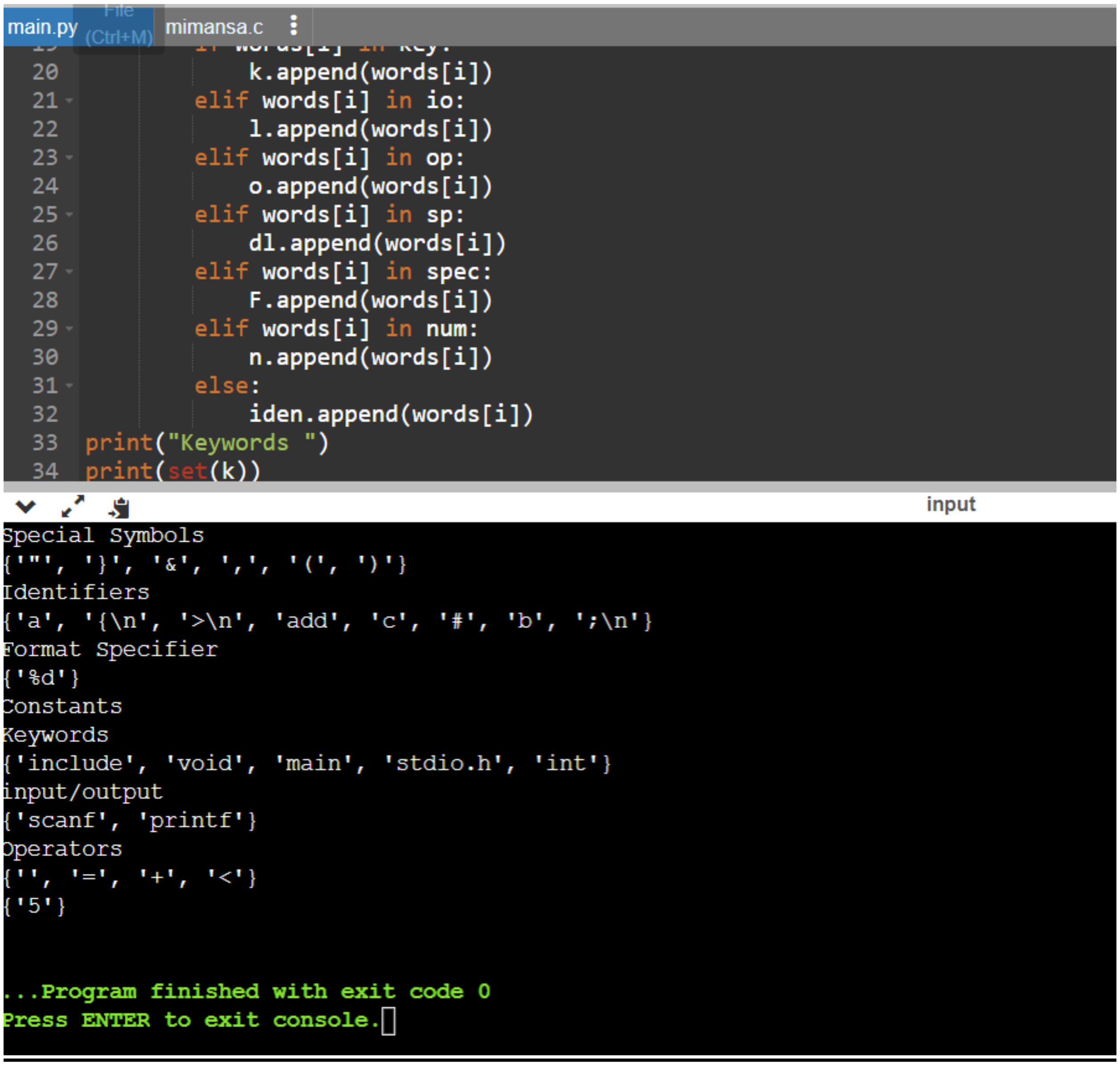
```
f=open("mimansa.c")
key=['int','float','string','include','stdio.h','char','break','if','else','switch','return','void'
,'while','struct','for', 'main']
iden=[]
sp={"(",")","{","}",";","&","",","}
spec=["%d","%f","%c","%s"]
num="012345678910"
n=[]
```

```
k=[]
o=[]
l=[]
io=['scanf','printf','cin','cout']
op="+-%*=/^><"
dl=[]
F=[]
for lines in f:
    words=lines.split(" ")
    for i in range(len(words)):
        if words[i] in key:
            k.append(words[i])
        elif words[i] in io:
            l.append(words[i])
        elif words[i] in op:
            o.append(words[i])
        elif words[i] in sp:
            dl.append(words[i])
        elif words[i] in spec:
            F.append(words[i])
        elif words[i] in num:
            n.append(words[i])
        else:
            iden.append(words[i])
print("Keywords ")
print(set(k))
print("input/output ")
print(set(l))
print("Operators ")
print(set(o))
print("Special Symbols ")
print(set(dl))
print("Identifiers" )
print(set(iden))
print("Format Specifier")
print(set(F))
print("Constants")
print(set(n))
```

C File:

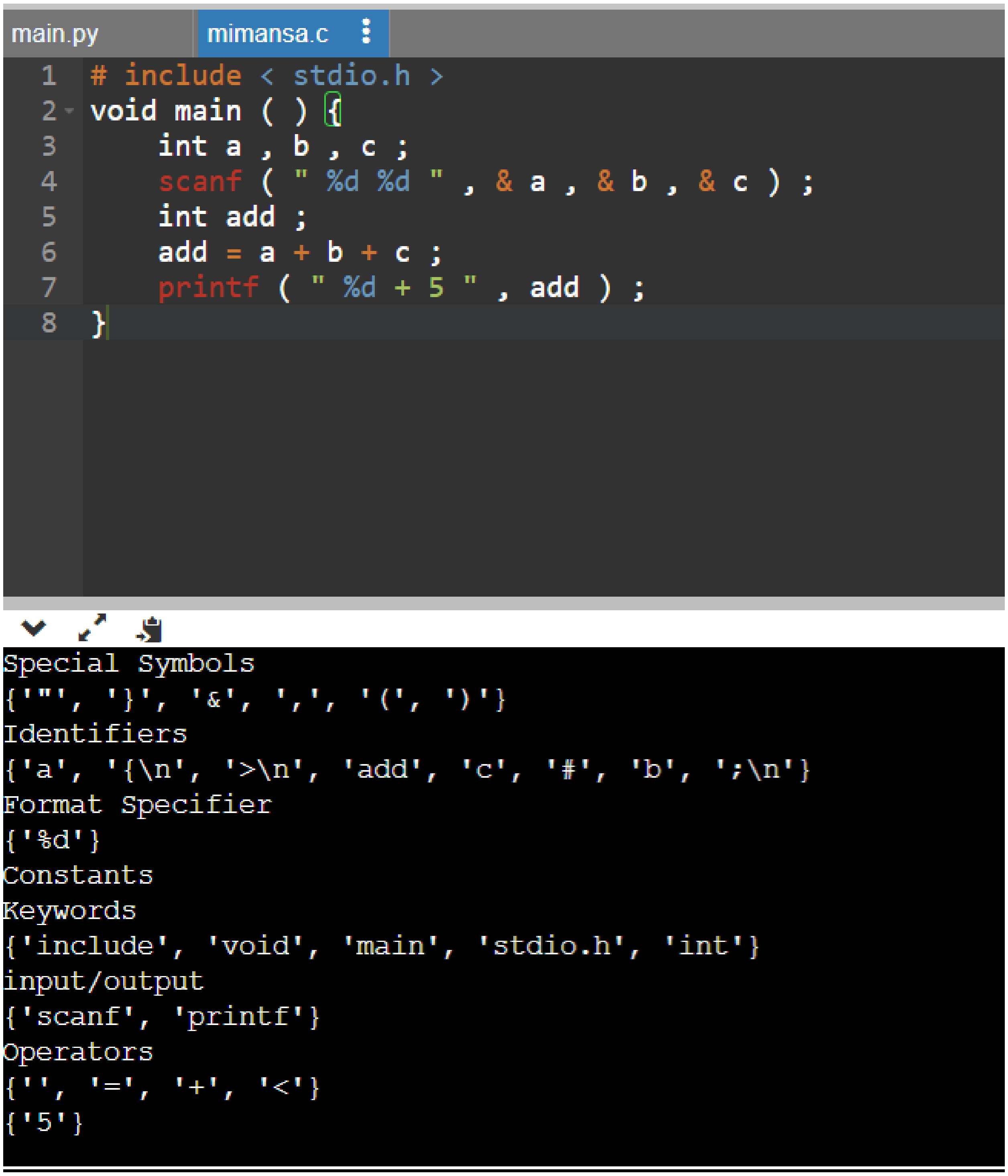
```
# include < stdio.h >
void main ( ) {
    int a , b , c ;
    scanf ( " %d %d " , & a , & b , & c ) ;
    int add ;
    add = a + b + c ;
    printf ( " %d + 5 " , add ) ;
}
```

Output Screenshots:



```
                k.append(words[i])
            elif words[i] in io:
                l.append(words[i])
            elif words[i] in op:
                o.append(words[i])
            elif words[i] in sp:
                dl.append(words[i])
            elif words[i] in spec:
                F.append(words[i])
            elif words[i] in num:
                n.append(words[i])
            else:
                iden.append(words[i])
print("Keywords ")
print(set(k))
```

```
Special Symbols
{'"', '}', '&', ',', '(', ')'}
Identifiers
{'a', '{\n', '>\n', 'add', 'c', '#', 'b', ';\n'}
Format Specifier
{'%d'}
Constants
Keywords
{'include', 'void', 'main', 'stdio.h', 'int'}
input/output
{'scanf', 'printf'}
Operators
{'', '=', '+', '<'}
{'5'}


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



```c
# include < stdio.h >
void main ( ) {
    int a , b , c ;
    scanf ( " %d %d " , & a , & b , & c ) ;
    int add ;
    add = a + b + c ;
    printf ( " %d + 5 " , add ) ;
}
```

```
Special Symbols
{'"', '}', '&', ',', '(', ')'}
Identifiers
{'a', '{\n', '>\n', 'add', 'c', '#', 'b', ';\n'}
Format Specifier
{'%d'}
Constants
Keywords
{'include', 'void', 'main', 'stdio.h', 'int'}
input/output
{'scanf', 'printf'}
Operators
{'', '=', '+', '<'}
{'5'}
```

Result:
Hence The lexical analyzer was implemented.

Date: 3rd Feb 2022

Exercise 2

Conversion of Regular expression to NFA

Aim:

To study and execute the program to convert a 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

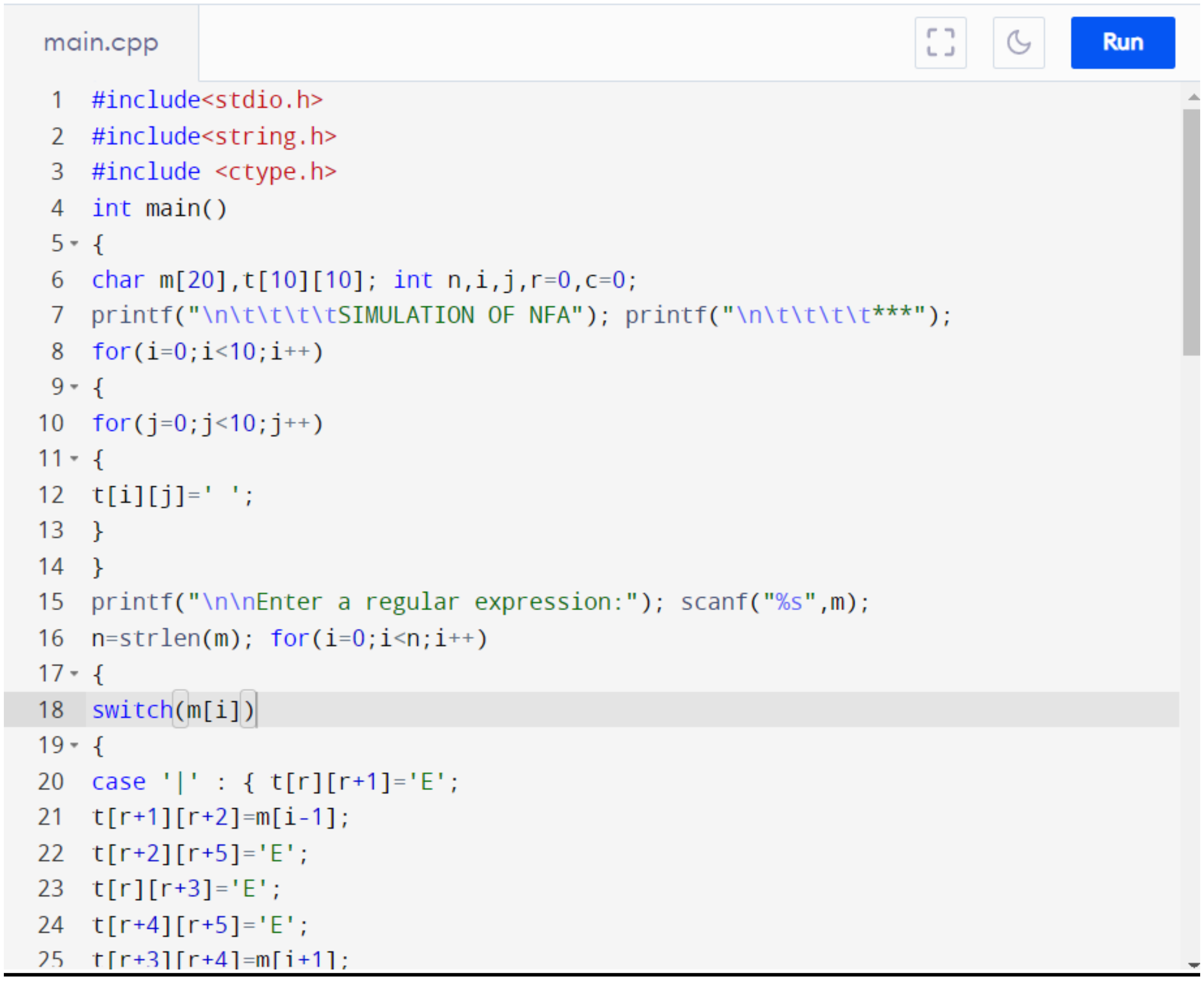
9. Stop

Program :

```
#include<stdio.h>
#include<string.h>
#include <ctype.h>
int main()
{
char m[20],t[10][10]; int n,i,j,r=0,c=0;
printf("\n\t\t\t\tSIMULATION OF NFA"); printf("\n\t\t\t\t***");
for(i=0;i<10;i++)
{
for(j=0;j<10;j++)
{
t[i][j]=' ';
}
}
printf("\n\nEnter a regular expression:"); scanf("%s",m);
```

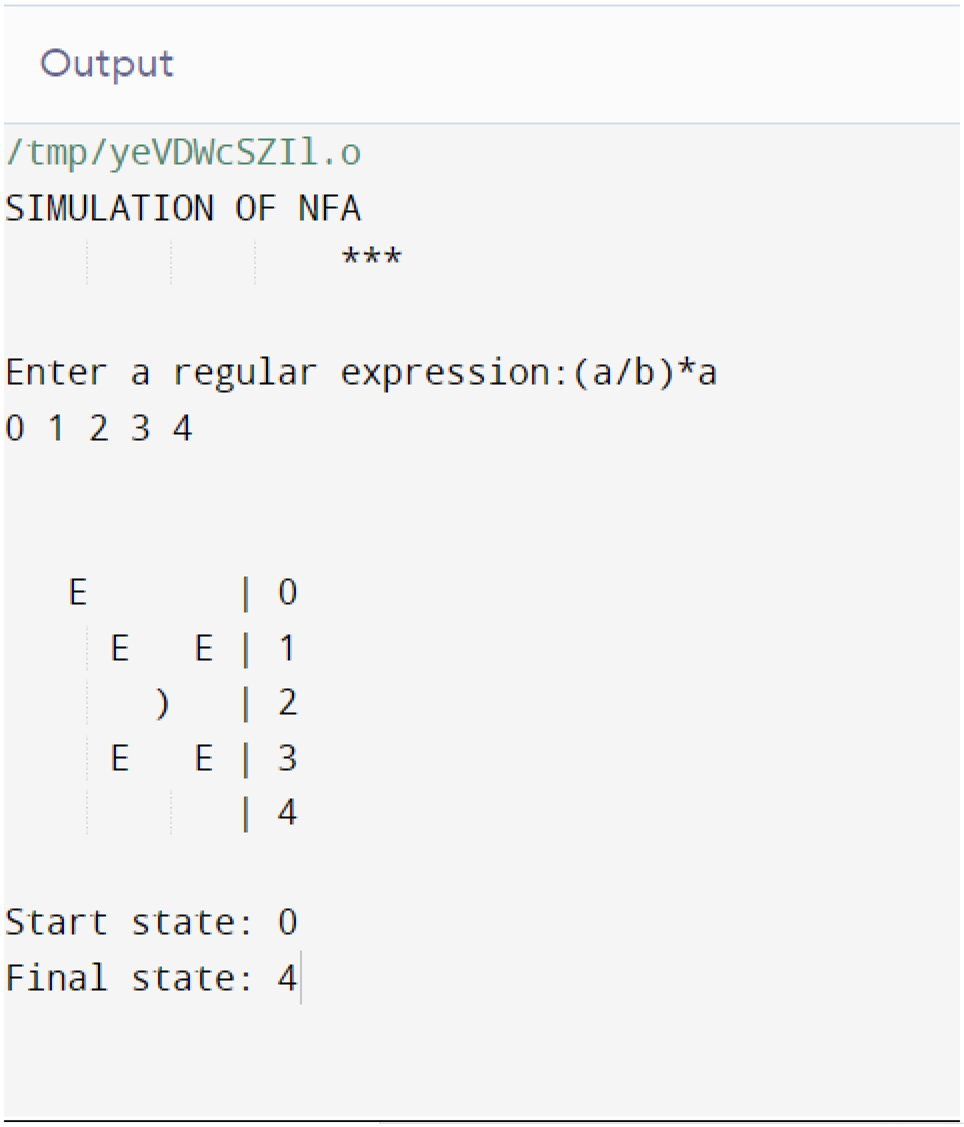
```
n=strlen(m); for(i=0;i<n;i++)
{
switch(m[i])
{
case '|' : { t[r][r+1]='E';
t[r+1][r+2]=m[i-1];
t[r+2][r+5]='E';
t[r][r+3]='E';
t[r+4][r+5]='E';
t[r+3][r+4]=m[i+1];
r=r+5; break;
}
case '*':{ t[r-1][r]='E';

t[r][r+1]='E';
t[r][r+3]='E';
t[r+1][r+2]=m[i-1];
t[r+2][r+1]='E';
t[r+2][r+3]='E';
r=r+3; break;
}
case '+': { t[r][r+1]=m[i-1];
t[r+1][r]='E';
r=r+1; break;
}
default:
{
if(c==0)
{
if((isalpha(m[i]))&&(isalpha(m[i+1])))
{
t[r][r+1]=m[i];
t[r+1][r+2]=m[i+1];
r=r+2; c=1;
} c=1;
}
else if(c==1)
{
if(isalpha(m[i+1]))
{ t[r][r+1]=m[i+1];
r=r+1; c=2;
}
}
else
{
if(isalpha(m[i+1]))
{ t[r][r+1]=m[i+1];
r=r+1; c=3;
}

}
}
break;
}
}
printf("\n"); for(j=0;j<=r;j++) printf(" %d",j);
printf("\n          \n");
printf("\n"); for(i=0;i<=r;i++)
{
for(j=0;j<=r;j++)
{
printf(" %c",t[i][j]);
}
printf(" | %d",i);
printf("\n");
}
printf("\nStart state: 0\nFinal state: %d",i-1);

return 0;
}
```

Output Screenshots:


main.cpp

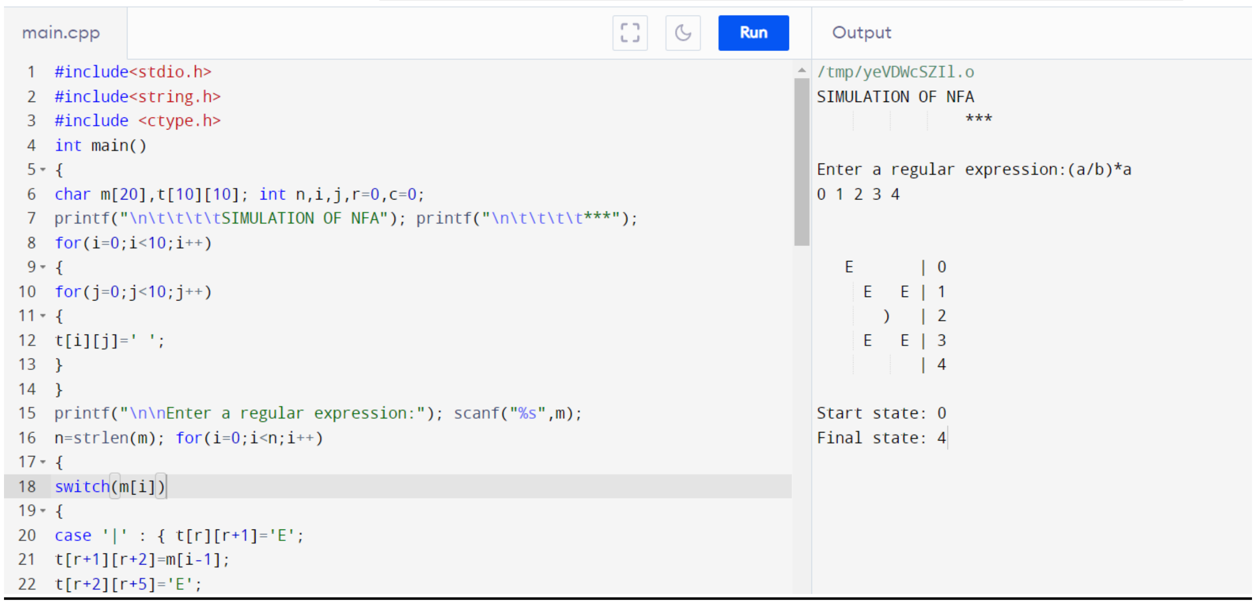
Run

```
#include<stdio.h>
#include<string.h>
#include <ctype.h>
int main()
{
char m[20],t[10][10]; int n,i,j,r=0,c=0;
printf("\n\t\t\t\tSIMULATION OF NFA"); printf("\n\t\t\t\t***");
for(i=0;i<10;i++)
{
for(j=0;j<10;j++)
{
t[i][j]=' ';
}
}
printf("\n\nEnter a regular expression:"); scanf("%s",m);
n=strlen(m); for(i=0;i<n;i++)
{
switch(m[i])
{
case '|' : { t[r][r+1]='E';
t[r+1][r+2]=m[i-1];
t[r+2][r+5]='E';
t[r][r+3]='E';
t[r+4][r+5]='E';
t[r+3][r+4]=m[i+1];
```



Output

```
/tmp/yeVDWcSZIl.o
SIMULATION OF NFA
                    ***


Enter a regular expression:(a/b)*a
0 1 2 3 4



   E        | 0
     E    E | 1
       )    | 2
     E    E | 3
            | 4

Start state: 0
Final state: 4
```



main.cpp

```
#include<stdio.h>
#include<string.h>
#include <ctype.h>
int main()
{
char m[20],t[10][10]; int n,i,j,r=0,c=0;
printf("\n\t\t\t\tSIMULATION OF NFA"); printf("\n\t\t\t\t***");
for(i=0;i<10;i++)
{
for(j=0;j<10;j++)
{
t[i][j]=' ';
}
}
printf("\n\nEnter a regular expression:"); scanf("%s",m);
n=strlen(m); for(i=0;i<n;i++)
{
switch(m[i])
{
case '|' : { t[r][r+1]='E';
t[r+1][r+2]=m[i-1];
t[r+2][r+5]='E';
```

Output

```
/tmp/yeVDWcSZIl.o
SIMULATION OF NFA
                    ***

Enter a regular expression:(a/b)*a
0 1 2 3 4


   E        | 0
     E    E | 1
       )    | 2
     E    E | 3
            | 4

Start state: 0
Final state: 4
```

Result:

Hence The RE was Converted to a NFA.

Date: 10th Feb 2022

Exercise 3

Conversion of RE to NFA to DFA

Aim:

To write a program to study and execute the program to convert a regular expression to NFA to DFA.

Algorithm:

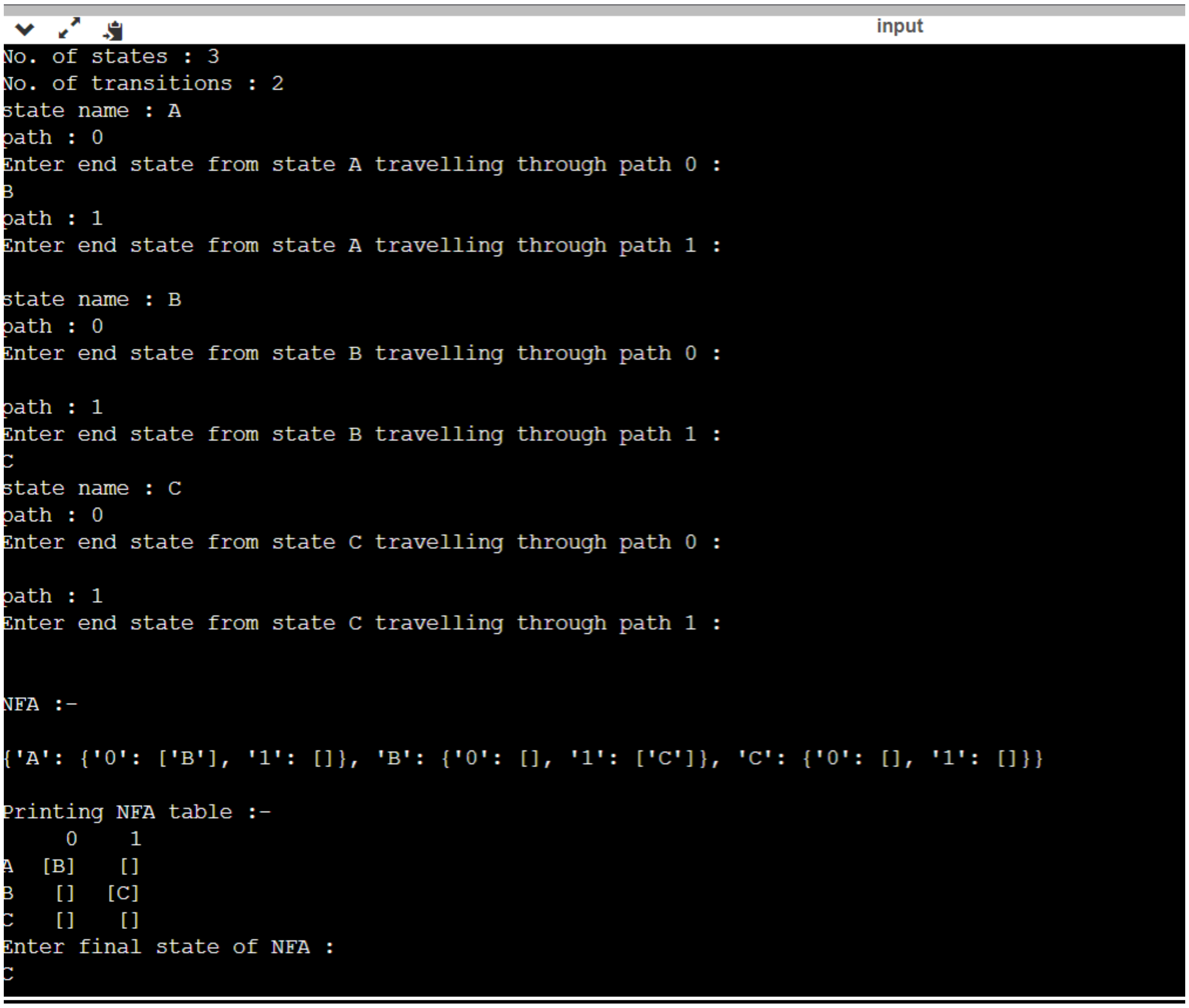
```
1. Start

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 :

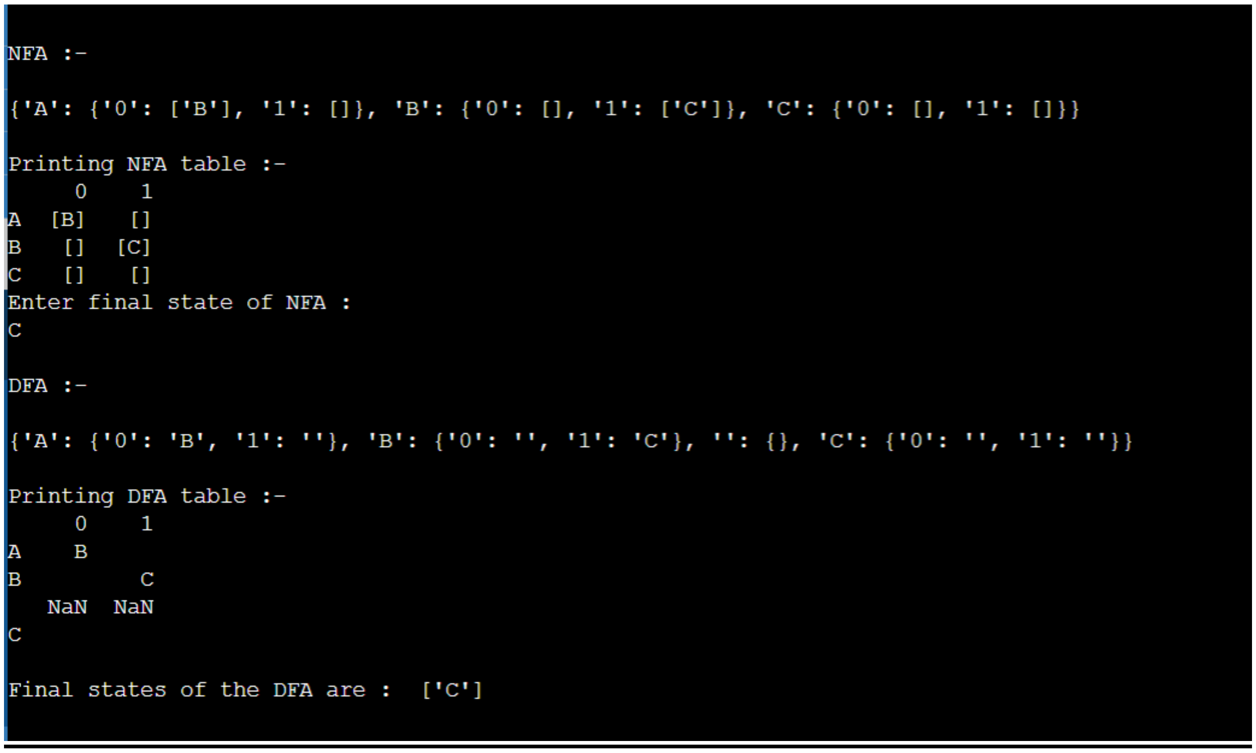
```
import pandas as pd

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)
```

Output Screenshots:



```
No. of states : 3
No. of transitions : 2
state name : A
path : 0
Enter end state from state A travelling through path 0 :
B
path : 1
Enter end state from state A travelling through path 1 :

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

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

Printing NFA table :-
     0    1
A  [B]   []
B   []  [C]
C   []   []
Enter final state of NFA :
C
```



```
NFA :-

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

Printing NFA table :-
     0    1
A  [B]   []
B   []  [C]
C   []   []
Enter final state of NFA :
C

DFA :-

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

Printing DFA table :-
     0    1
A    B
B         C
   NaN  NaN
C

Final states of the DFA are :  ['C']
```

Result:

The RE was Converted to a NFA which was converted to DFA.

Date: 17th Feb 2022
Exercise 4

Implementation of Syntax Analysis:Conversion of Infix to Postfix and Prefix

Aim:

To write a program to study and execute the syntax analysis program to convert an infix expression to postfix and prefix

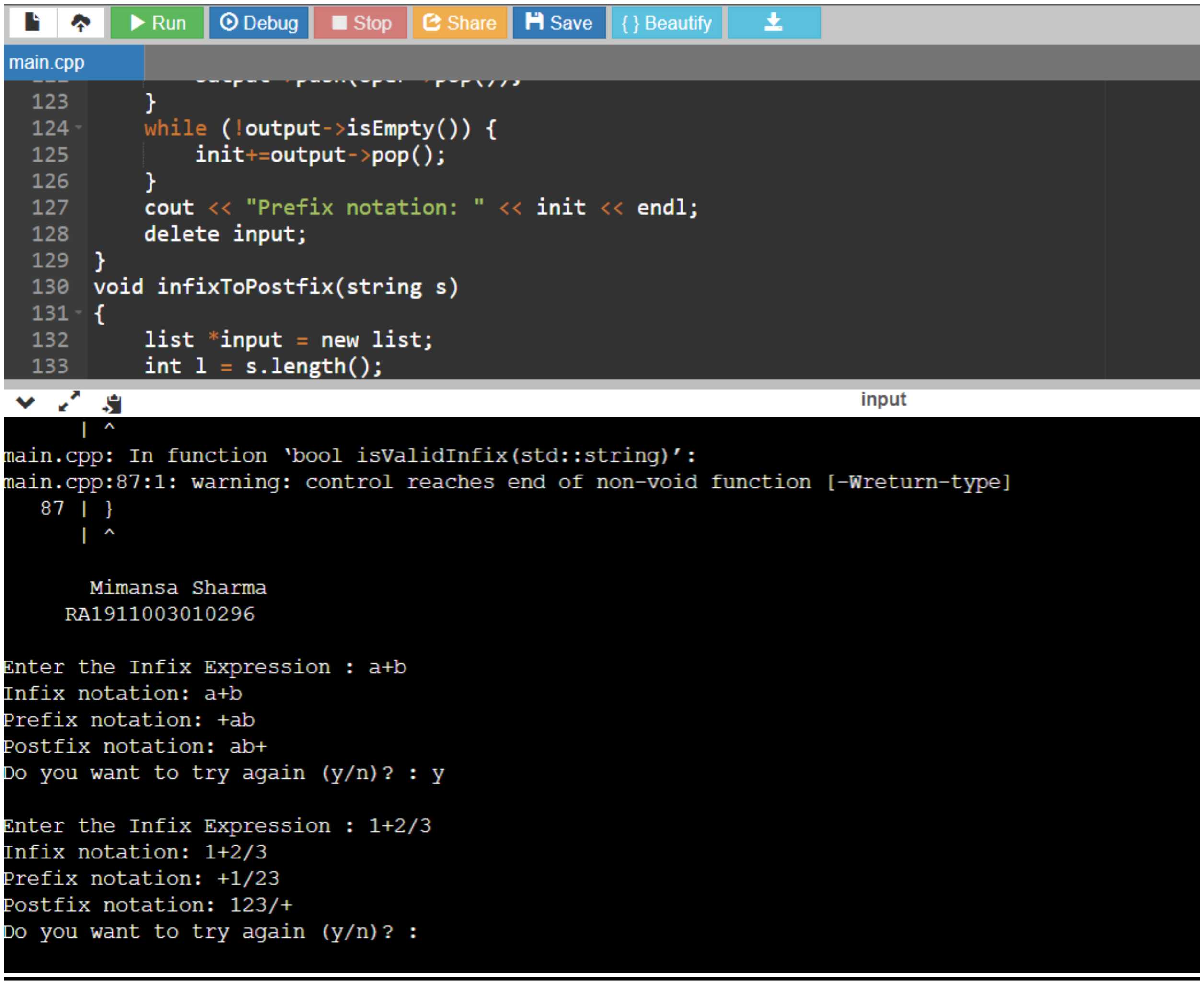
Algorithm:

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 :

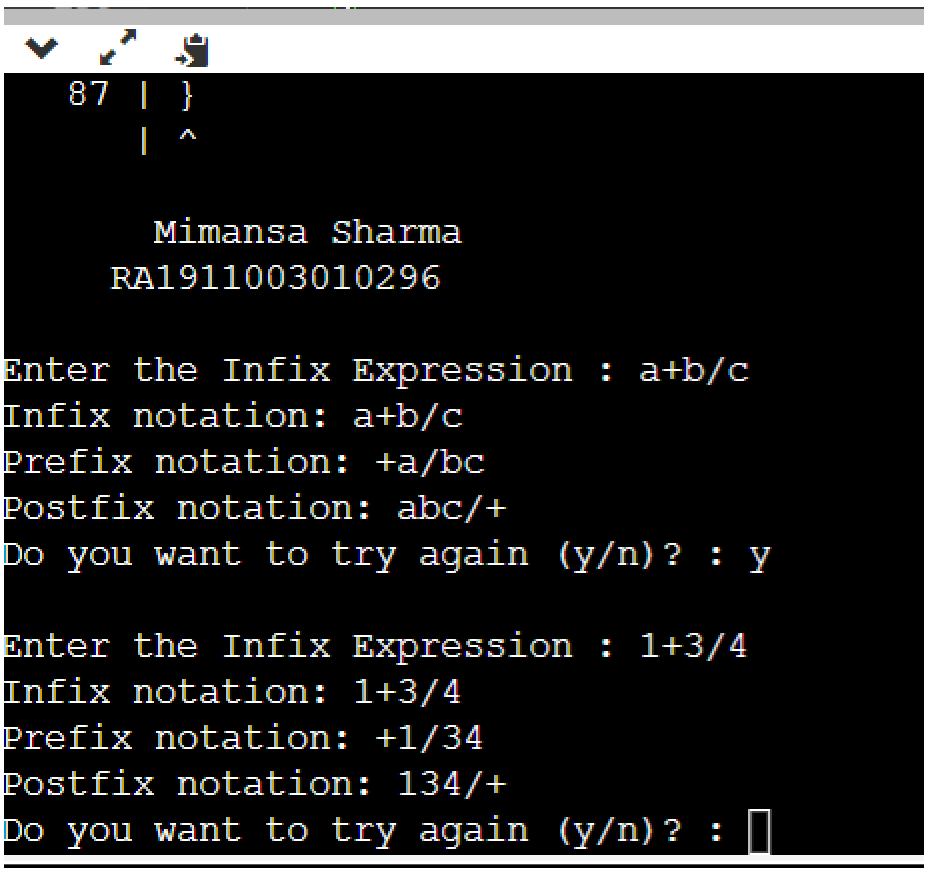
```
   #include <iostream>
using namespace std;
int submain()
{
        cout<<"\n      Mimansa Sharma\n";
        cout<<"    RA1911003010296\n";
}
struct node {
    char data;
    node *next;
};
class list
{
private:
    node *top;
public:
    list() { top=NULL; }
    ~list() { while (!isEmpty()) {
            pop();
        }
    }
    char peek();
    bool isEmpty();
    void push(char new_data);
    char pop();
};
bool list::isEmpty() {
    return (top==NULL);
}
char list::peek() {
    if (!isEmpty()) {
        char value = top->data;
        return value;
    } else {
        cout << "Stack is empty!!" << endl;
        exit(1);
    }
}
void list::push(char new_data) {
    node *temp = new node;
    temp->data=new_data;
    temp->next=top;
    top=temp;
}
char list::pop() {
    if (!isEmpty()) {
        char value=top->data;
        node *oldtop = top;
        top=top->next;
        delete oldtop;
        return value;
    } else {
        cout << "You cannot pop an empty stack!" << endl;
        exit(1);
    }
}
int prec(char c)
{
    if(c == '^')
        return 3;
    else if(c == '*' || c == '/')
        return 2;
    else if(c == '+' || c == '-')
        return 1;
    else if(c=='#')
        return -2;
    else
        return -1;
}
bool isOperand(char s) {
    return (s>='a'&&s<='z'||s>='A'&&s<='Z'||s>='0'&&s<='9');
}
bool isOperator(char s) {
    return (s=='+'||s=='/'||s=='-'||s=='*'||s=='^');
}
bool isValidChar (char s) {
    return (isOperator(s)||isOperand(s)||s=='('||s==')');
}
bool isValidInfix (string s) {
    int l = s.length();
    for(int i=0;i<l;i++) {
        if (!isValidChar(s[i])) {
            return false;
            break;
        }
    }
}
void infixToPrefix(string s)
{
    list *input = new list;
    int l = s.length();
    string init;
    for(int i=0;i<l;i++) {
        input->push(s[i]);
    }
    list *output = new list;
    list *oper = new list;
    while(!input->isEmpty()) {
        char val = input->pop();
        if (isOperand(val)) {
            output->push(val);
        } else if (val==')') {
            oper->push(val);
        } else if(isOperator(val)) {
            bool pushed=false;
            while(!pushed) {
                if (oper->isEmpty()||oper->peek()==')'||prec(val)>=prec(oper->peek())) {
                    oper->push(val);
                    pushed=true;
                } else {
                    output->push(oper->pop());
                }
            }
        } else if (val=='(') {
            while (oper->peek()!=')') {
                output->push(oper->pop());
            }
            oper->pop();
        }
    }
    while(!oper->isEmpty()) {
        output->push(oper->pop());
    }
    while (!output->isEmpty()) {
        init+=output->pop();
    }
    cout << "Prefix notation: " << init << endl;
    delete input;
}
void infixToPostfix(string s)
{
    list *input = new list;
    int l = s.length();
    string ns;
    for(int i = 0; i < l; i++)
    {
        if (isOperand(s[i])) {
            ns+=s[i];
        }
        else if(s[i] == '(') {
            input->push('(');
        }
        else if(s[i] == ')')
        {
            while(!input->isEmpty()&&input->peek()!='(')
            {
                ns+=input->pop();
            }
            if(input->peek()=='(')
            {
                input->pop();
            }
        }
        else{
            while(!input->isEmpty()&&prec(s[i])<=prec(input->peek()))
            {
                char c = input->peek();
                input->pop();
                ns += c;
            }
            input->push(s[i]);
        }
    }
    while(!input->isEmpty())
    {
        char c = input->pop();
        if (c!='(' && c!=')') {
            ns+=c;
        }
    }
    cout << "Postfix notation: " << ns << endl;
    delete input;
}
int main( ) {
            submain();
    char Infix_expression[100];
    char tryAgain;
    do {
        cout<<"\nEnter the Infix Expression : ";
        cin>>Infix_expression;
        cout << "Infix notation: " << Infix_expression << endl;
        if (isValidInfix(Infix_expression)) {
            infixToPrefix(Infix_expression);
            infixToPostfix(Infix_expression);
        } else {
            cout << "Infix variable ranges from A to Z either in lower or upper case only " << endl;
            cout << "Please check your infix expression" << endl;
        }
        cout<<"Do you want to try again (y/n)? : ";
        cin>>tryAgain;
        tryAgain=tolower(tryAgain);
    }
    while (tryAgain!='n');
    return 0;
}
```

Output Screenshots:



```
    }
    while (!output->isEmpty()) {
        init+=output->pop();
    }
    cout << "Prefix notation: " << init << endl;
    delete input;
}
void infixToPostfix(string s)
{
    list *input = new list;
    int l = s.length();
```

```
      | ^
main.cpp: In function ‘bool isValidInfix(std::string)’:
main.cpp:87:1: warning: control reaches end of non-void function [-Wreturn-type]
   87 | }
      | ^

       Mimansa Sharma
     RA1911003010296

Enter the Infix Expression : a+b
Infix notation: a+b
Prefix notation: +ab
Postfix notation: ab+
Do you want to try again (y/n)? : y

Enter the Infix Expression : 1+2/3
Infix notation: 1+2/3
Prefix notation: +1/23
Postfix notation: 123/+
Do you want to try again (y/n)? :
```


```
   87 | }
      | ^

       Mimansa Sharma
     RA1911003010296

Enter the Infix Expression : a+b/c
Infix notation: a+b/c
Prefix notation: +a/bc
Postfix notation: abc/+
Do you want to try again (y/n)? : y

Enter the Infix Expression : 1+3/4
Infix notation: 1+3/4
Prefix notation: +1/34
Postfix notation: 134/+
Do you want to try again (y/n)? :
```

Result:
The syntax analysis was implemented and the infix expression was Converted to Postfix and Prefix expressions.

Date: 9th March 2022

Exercise 5

Computation of First and Follow

Aim:

To write a program to study and execute first and follow

Algorithm:

For computing the first:

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

Example: F ->1|1id

We can write it as FIRST(F) -> { (,1id)

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 -> ¢ then add € 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.

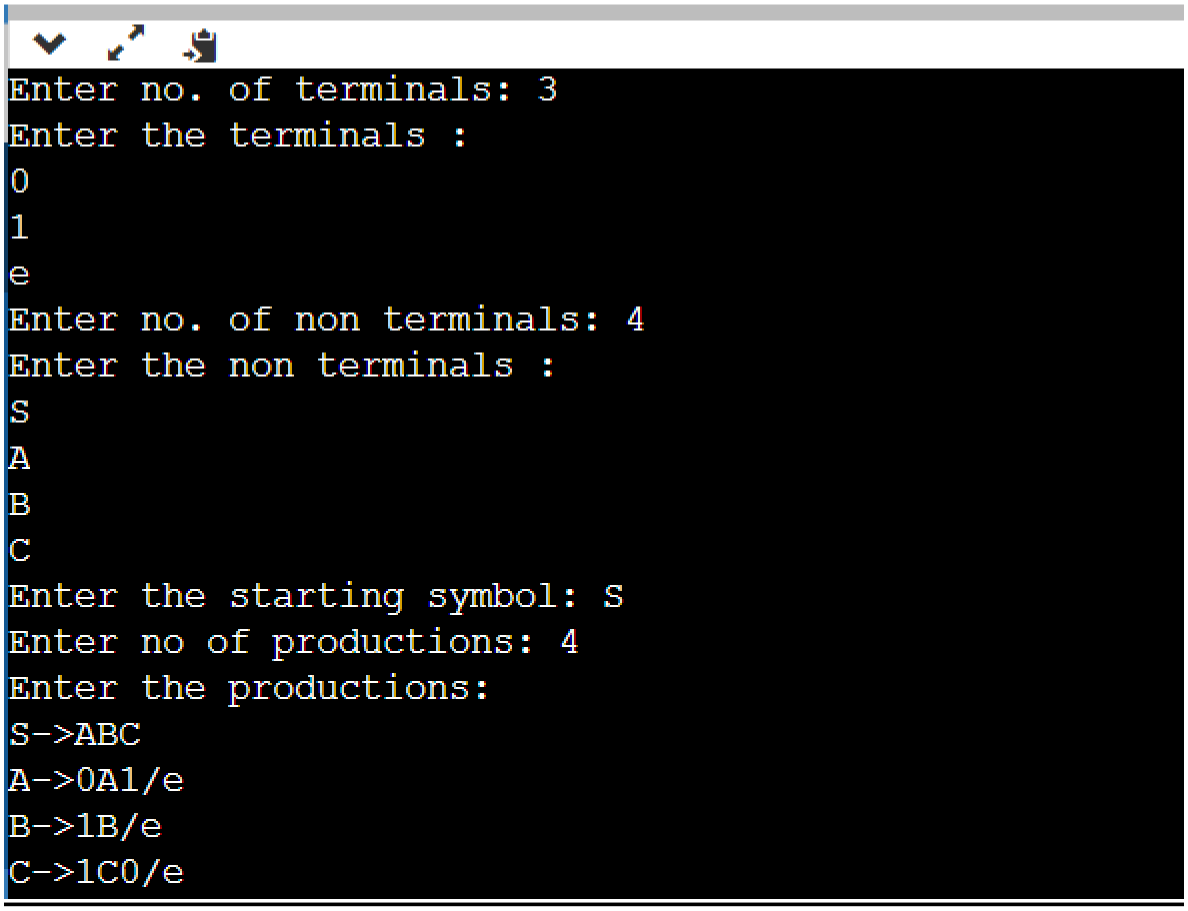
Program :

```
import sys
sys.setrecursionlimit(60)

def first(string):
    #print("first({})".format(string))
    first_ = set()
    if string in non_terminals:
        alternatives = productions_dict[string]

        for alternative in alternatives:
            first_2 = first(alternative)
            first_ = first_ |first_2

    elif string in terminals:
        first_ = {string}

    elif string=='' or string=='@':
        first_ = {'@'}

    else:
        first_2 = first(string[0])
        if '@' in first_2:
            i = 1
            while '@' in first_2:
                #print("inside while")

                first_ = first_ | (first_2 - {'@'})
                #print('string[i:]=', string[i:])
                if string[i:] in terminals:
                    first_ = first_ | {string[i:]}
                    break
                elif string[i:] == '':
                    first_ = first_ | {'@'}
                    break
                first_2 = first(string[i:])
                first_ = first_ | first_2 - {'@'}
                i += 1
        else:
            first_ = first_ | first_2


    #print("returning for first({})".format(string),first_)
    return  first_


def follow(nT):
    #print("inside follow({})".format(nT))
    follow_ = set()
    #print("FOLLOW", FOLLOW)
    prods = productions_dict.items()
    if nT==starting_symbol:
        follow_ = follow_ | {'$'}
    for nt,rhs in prods:
        #print("nt to rhs", nt,rhs)
        for alt in rhs:
            for char in alt:
                if char==nT:
                    following_str = alt[alt.index(char) + 1:]
                    if following_str=='':
                        if nt==nT:
                            continue
                        else:
                            follow_ = follow_ | follow(nt)
                    else:
                        follow_2 = first(following_str)
                        if '@' in follow_2:
                            follow_ = follow_ | follow_2-{'@'}
                            follow_ = follow_ | follow(nt)
                        else:
                            follow_ = follow_ | follow_2
    #print("returning for follow({})".format(nT),follow_)
    return follow_




no_of_terminals=int(input("Enter no. of terminals: "))

terminals = []

print("Enter the terminals :")
for _ in range(no_of_terminals):
    terminals.append(input())

no_of_non_terminals=int(input("Enter no. of non terminals: "))

non_terminals = []

print("Enter the non terminals :")
for _ in range(no_of_non_terminals):
    non_terminals.append(input())

starting_symbol = input("Enter the starting symbol: ")

no_of_productions = int(input("Enter no of productions: "))

productions = []
```

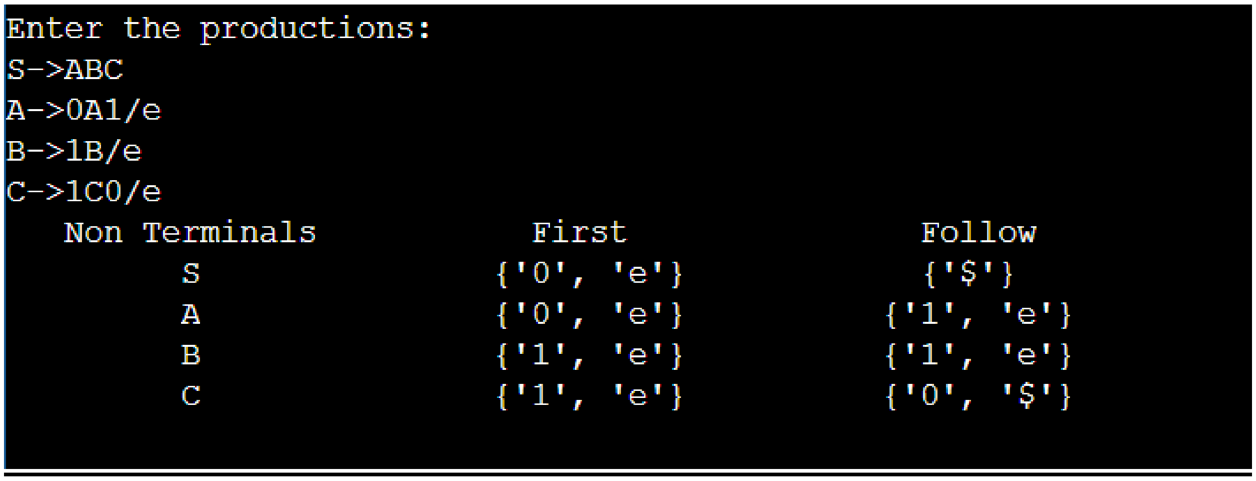
```
print("Enter the productions:")
for _ in range(no_of_productions):
    productions.append(input())


#print("terminals", terminals)

#print("non terminals", non_terminals)

#print("productions",productions)


productions_dict = {}

for nT in non_terminals:
    productions_dict[nT] = []


#print("productions_dict",productions_dict)

for production in productions:
    nonterm_to_prod = production.split("->")
    alternatives = nonterm_to_prod[1].split("/")
    for alternative in alternatives:
        productions_dict[nonterm_to_prod[0]].append(alternative)

#print("productions_dict",productions_dict)

#print("nonterm_to_prod",nonterm_to_prod)
#print("alternatives",alternatives)


FIRST = {}
FOLLOW = {}

for non_terminal in non_terminals:
    FIRST[non_terminal] = set()

for non_terminal in non_terminals:
    FOLLOW[non_terminal] = set()

#print("FIRST",FIRST)
```

```
for non_terminal in non_terminals:
    FIRST[non_terminal] = FIRST[non_terminal] | first(non_terminal)

#print("FIRST",FIRST)


FOLLOW[starting_symbol] = FOLLOW[starting_symbol] | {'$'}
for non_terminal in non_terminals:
    FOLLOW[non_terminal] = FOLLOW[non_terminal] | follow(non_terminal)

#print("FOLLOW", FOLLOW)

print("{: ^20}{: ^20}{: ^20}".format('Non Terminals','First','Follow'))
for non_terminal in non_terminals:
    print("{: ^20}{: ^20}{:
^20}".format(non_terminal,str(FIRST[non_terminal]),str(FOLLOW[non_terminal])))
```

<u>Output Screenshots:</u>



```
Enter no. of terminals: 3
Enter the terminals :
0
1
e
Enter no. of non terminals: 4
Enter the non terminals :
S
A
B
C
Enter the starting symbol: S
Enter no of productions: 4
Enter the productions:
S->ABC
A->0A1/e
B->1B/e
C->1C0/e
```



```
Enter the productions:
S->ABC
A->0A1/e
B->1B/e
C->1C0/e
   Non Terminals            First                Follow
         S              {'0', 'e'}               {'$'}
         A              {'0', 'e'}             {'1', 'e'}
         B              {'1', 'e'}             {'1', 'e'}
         C              {'1', 'e'}             {'0', '$'}
```

Result:

Hence First and Follow was implemented successfully

Date: 10th March 2022

Exercise 6

Elimination of Left Factoring and Left Recursion

Aim:

To study and execute elimination of left factoring and left recursion

Algorithm:

LEFT RECURSION:

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->Acl|Aa2|.....|Aom
A->B1[B2/.... | pn
Then replace it by
A->BiA'=1.2.3,....m
A->awjA'j=1.23,....n
A->E

6. After eliminating the left recursion by applying these rules, display the productions without left recursion.
7. Stop.

LEFT FACTORING:

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->aBl/aB2

4. If found, replace the particular productions with:

A->aA'
A'->Bl | B2Je

5. Display the output
6. Exit

Program :

LEFT RECURSION:

```
//left recursion 296
```

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
int n, j, l, i, k;
int length[10] = {};
string d, a, b, flag;
char c;
cout<<"Enter Parent Non-Terminal: ";
cin >> c;
d.push_back(c);
a += d + "\'->";
d += "->";
b += d;
cout<<"Enter productions: ";
cin >> n;
for (int i = 0; i < n; i++)
{
cout<<"Enter Production ";
cout<<i + 1<<" :";
cin >> flag;
length[i] = flag.size();
d += flag;
if (i != n - 1)
{
d += "|";
}
}
cout<<"The Production Rule is: ";
cout<<d<<endl;
for (i = 0, k = 3; i < n; i++)
{
if (d[0] != d[k])
{
cout<<"Production: "<< i + 1;
cout<<" does not have left recursion.";
cout<<endl;
if (d[k] == '#')
{
b.push_back(d[0]);
b += "\'";
}
else
{
for (j = k; j < k + length[i]; j++)
{
b.push_back(d[j]);
}
k = j + 1;
b.push_back(d[0]);
b += "\'|";
}
}
else
{
cout<<"Production: "<< i + 1 ;
cout<< " has left recursion";
cout<< endl;
if (d[k] != '#')
{
for (l = k + 1; l < k + length[i]; l++)
{
a.push_back(d[l]);
}
k = l + 1;
a.push_back(d[0]);
a += "\'|";
}
}
}
a += "#";
cout << b << endl;
cout << a << endl;
return 0;
}
```

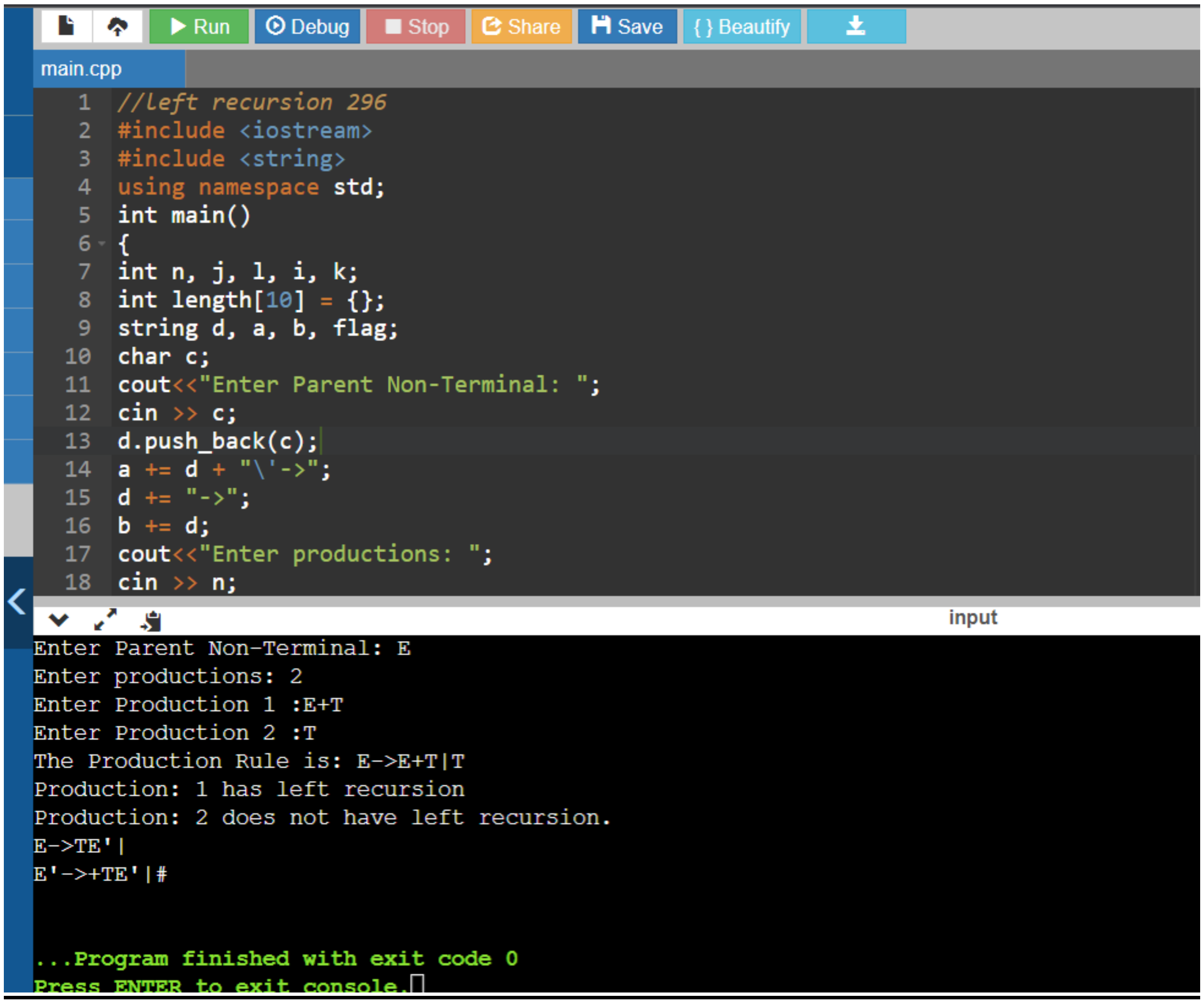
LEFT FACTORING:

```
def leftFactoring(s):
    k=[]
    l=[]
    n=""
    k=s.split('->')
    l=k[1].split('|')
    for i in range(0,len(l)-1):
        for j in range(0,min(len(l[i]),len(l[i+1]))):
            if(l[i][j]==l[i+1][j]):
                if l[i][j] not in n:
```

```
                n=n+l[i][j]
    print(k[0]+'->'+n+"R")
    m=k[1].split(n)
    print("R->",end="")
    for i in range(1,len(m)):
        print(m[i],end="")

s=input("Enter the production: ") #main function
while(True):
    leftFactoring(s)
    print("\ndo you have another production?")
    T=input("y/n:")
    if T=='y':
        s=input("Enter the production: ")
    elif T=='n':
        break
```

Output Screenshots:

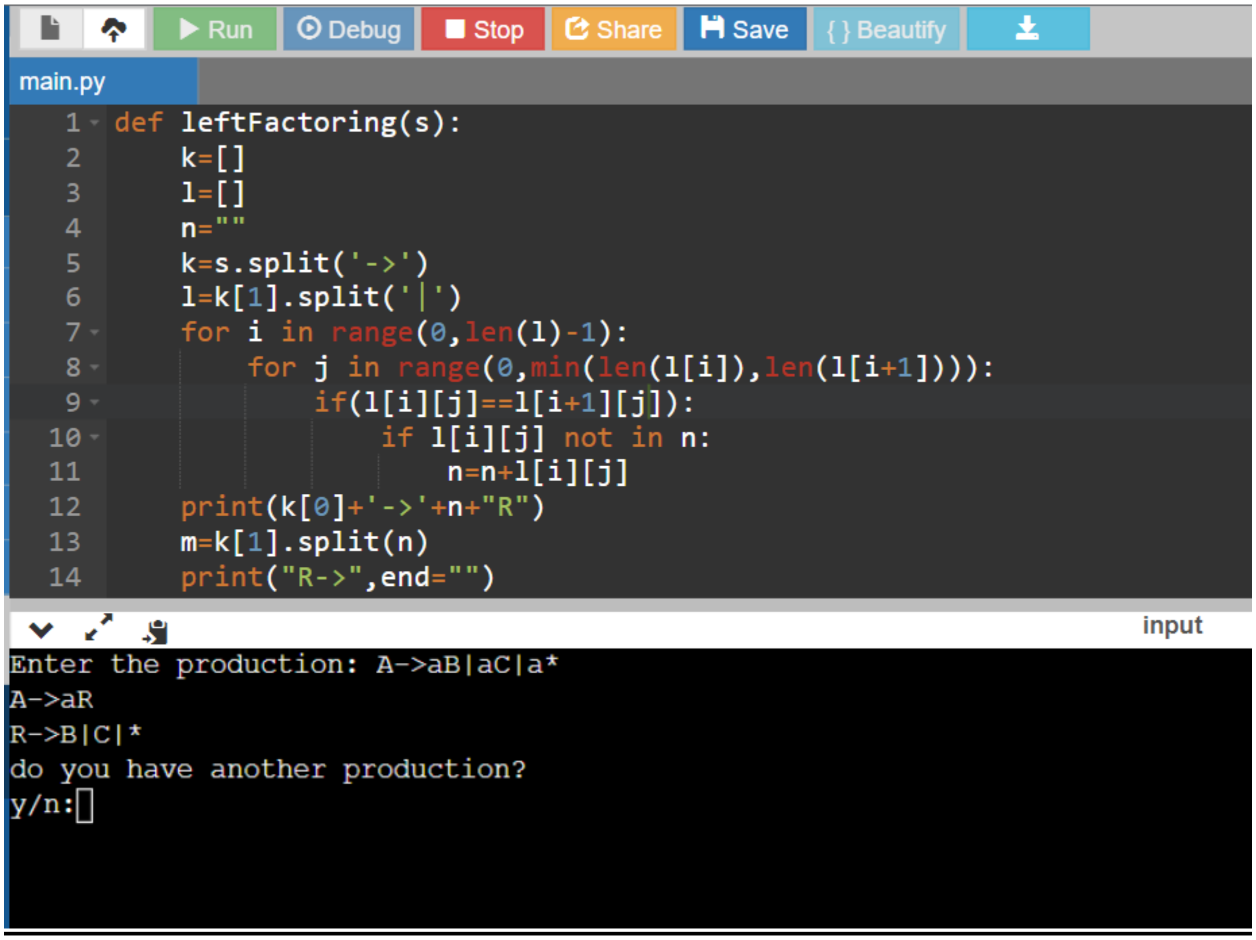
LEFT RECURSION:



```cpp
//left recursion 296
#include <iostream>
#include <string>
using namespace std;
int main()
{
int n, j, l, i, k;
int length[10] = {};
string d, a, b, flag;
char c;
cout<<"Enter Parent Non-Terminal: ";
cin >> c;
d.push_back(c);
a += d + "\'->";
d += "->";
b += d;
cout<<"Enter productions: ";
cin >> n;
```

```
Enter Parent Non-Terminal: E
Enter productions: 2
Enter Production 1 :E+T
Enter Production 2 :T
The Production Rule is: E->E+T|T
Production: 1 has left recursion
Production: 2 does not have left recursion.
E->TE'|
E'->+TE'|#


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

LEFT FACTORING:



```python
def leftFactoring(s):
    k=[]
    l=[]
    n=""
    k=s.split('->')
    l=k[1].split('|')
    for i in range(0,len(l)-1):
        for j in range(0,min(len(l[i]),len(l[i+1]))):
            if(l[i][j]==l[i+1][j]):
                if l[i][j] not in n:
                    n=n+l[i][j]
    print(k[0]+'->'+n+"R")
    m=k[1].split(n)
    print("R->",end="")
```

```
Enter the production: A->aB|aC|a*
A->aR
R->B|C|*
do you have another production?
y/n:
```

Result:

Hence Elimination of left factoring and left recursion was executed successfully.

Date: 23rd March 2022

Exercise 7

Implementing Predictive Parser

Aim:

To study and execute implementation of predictive parser

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 — ain G) {
for (each terminal a in FIRST(a)) |
add A — a to M[A, a];
if (¢ is in FIRST(a))
for (each symbol b in FOLLOW(A))
add A — ato M[A, b];
5. Print the resulting stack. |
6. Print if the grammar is accepted or not.
7. Exit the program.
```

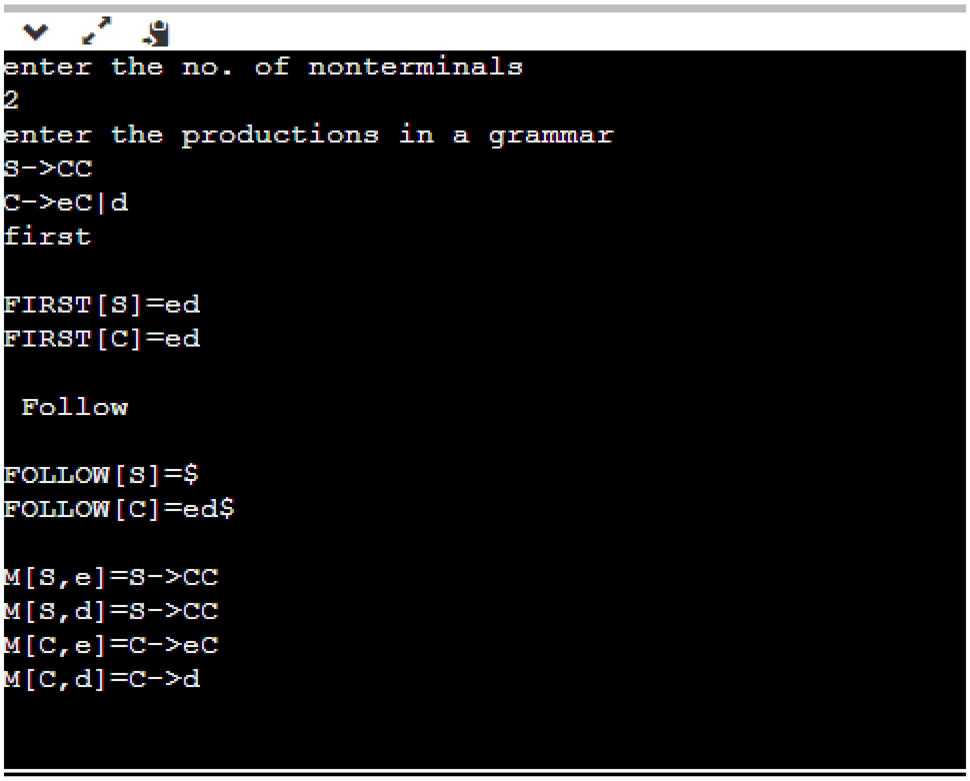
Program :

```
#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;

   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\n");
for(i=0;i<n;i++)
    printf("FIRST[%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("\n Follow \n\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],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++;}}}
```

Output Screenshots:



```
enter the no. of nonterminals
2
enter the productions in a grammar
S->CC
C->eC|d
first

FIRST[S]=ed
FIRST[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:

Predictive Parser was executed and its Implementation was successful

Date: 31st March 2022

Exercise 8

Implementing Shift Reduce Parser

Aim:

To write a program to study and execute implementation of shift reduce parser

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';
        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;
          }
     }
}
```

Output Screenshots:

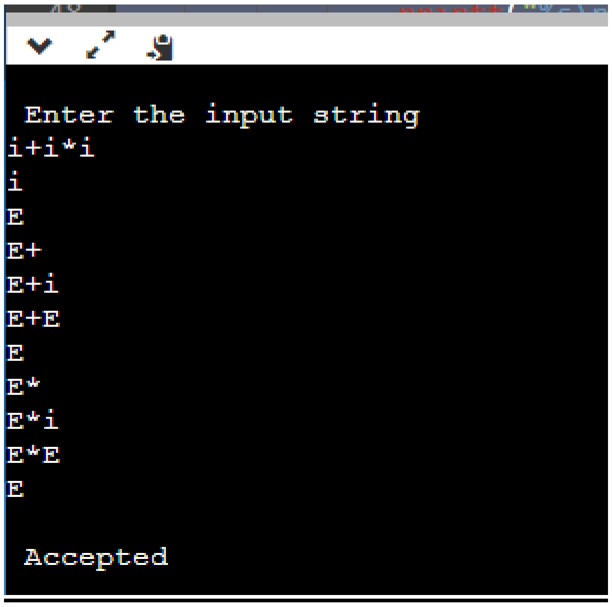
Input:

E->E+E

E->E*E

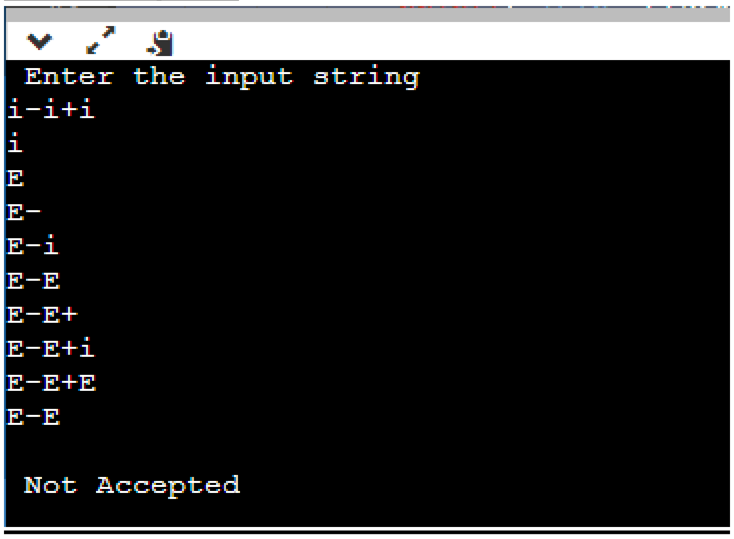
E->i

Accepted:



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

 Accepted
```

Not Accepted:



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

 Not Accepted
```

Result:

Hence Shift Reduce Parser was executed and its Implementation was successful

Date: 8th April 2022

Exercise 9

Implementing Leading and Trailing

Aim:

To write a program to study and calculate leading and trailing for the given grammar

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

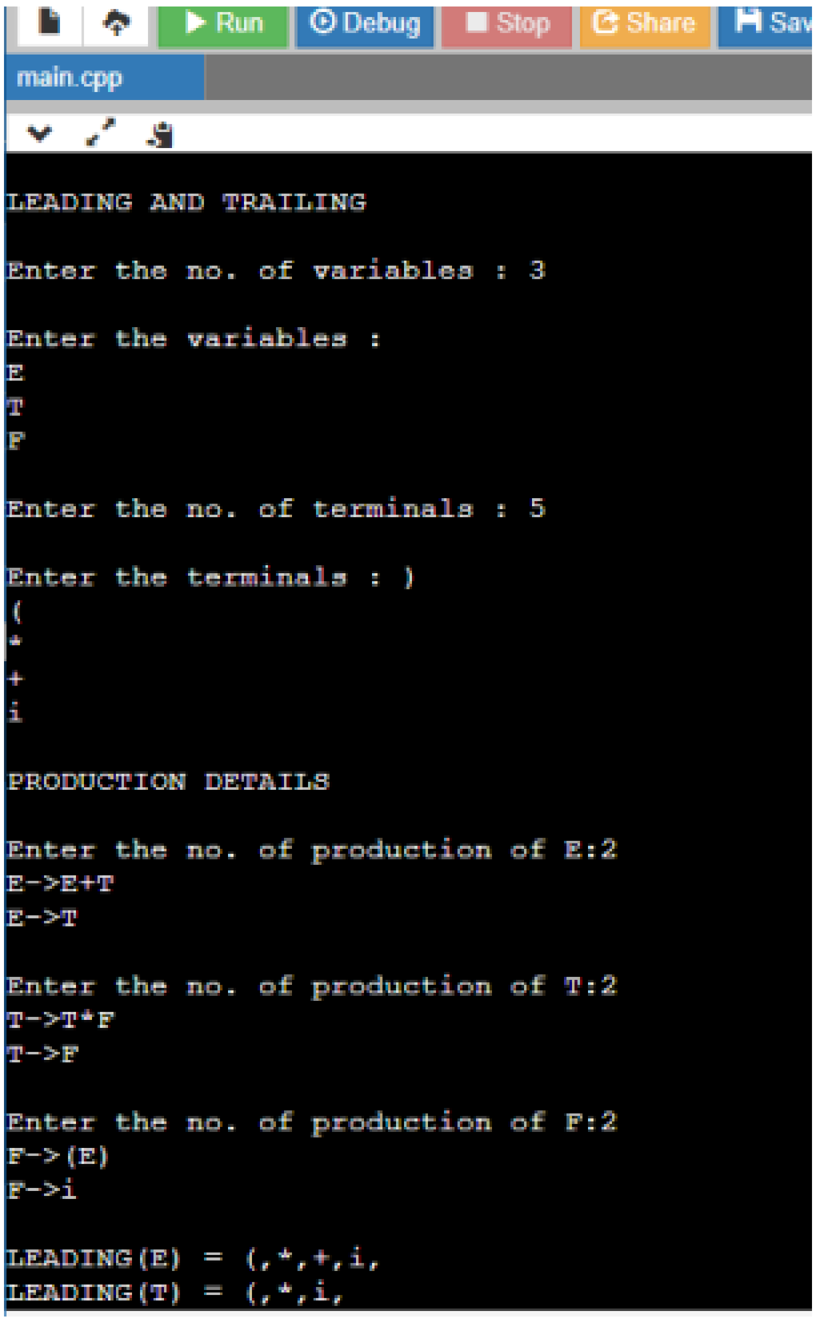
Program :

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
using namespace std;

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<<"\nLEADING 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<<"\nPRODUCTION 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++)
                        {
```

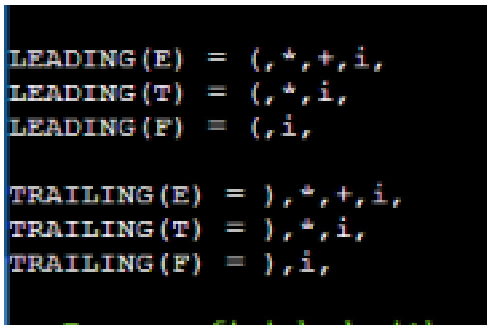
```
                                        for(m=1;m<vars;m++)
                                        {
                                                if(gram[i].rhs[j][0]==var[m])
                                                {
                                                        temp=m;
                                                        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]<<",";
                        }
                }
        }
        int main()
        {

                get();
                leading();
                trailing();
                display();
        }
```

Output Screenshots:



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

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

Result:

Leading and trailing was calculated and its Implementation was successful

Date: 8th April 2022

Exercise 10

Implementing LR(0) Parser

Aim:

To write a program study and execute implementation of LR(0) parser

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>
#include<conio.h>
#include<string.h>

using namespace std;

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;
}

int main()
{
        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<<" ";

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(int 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";
```

```
        }
}
```

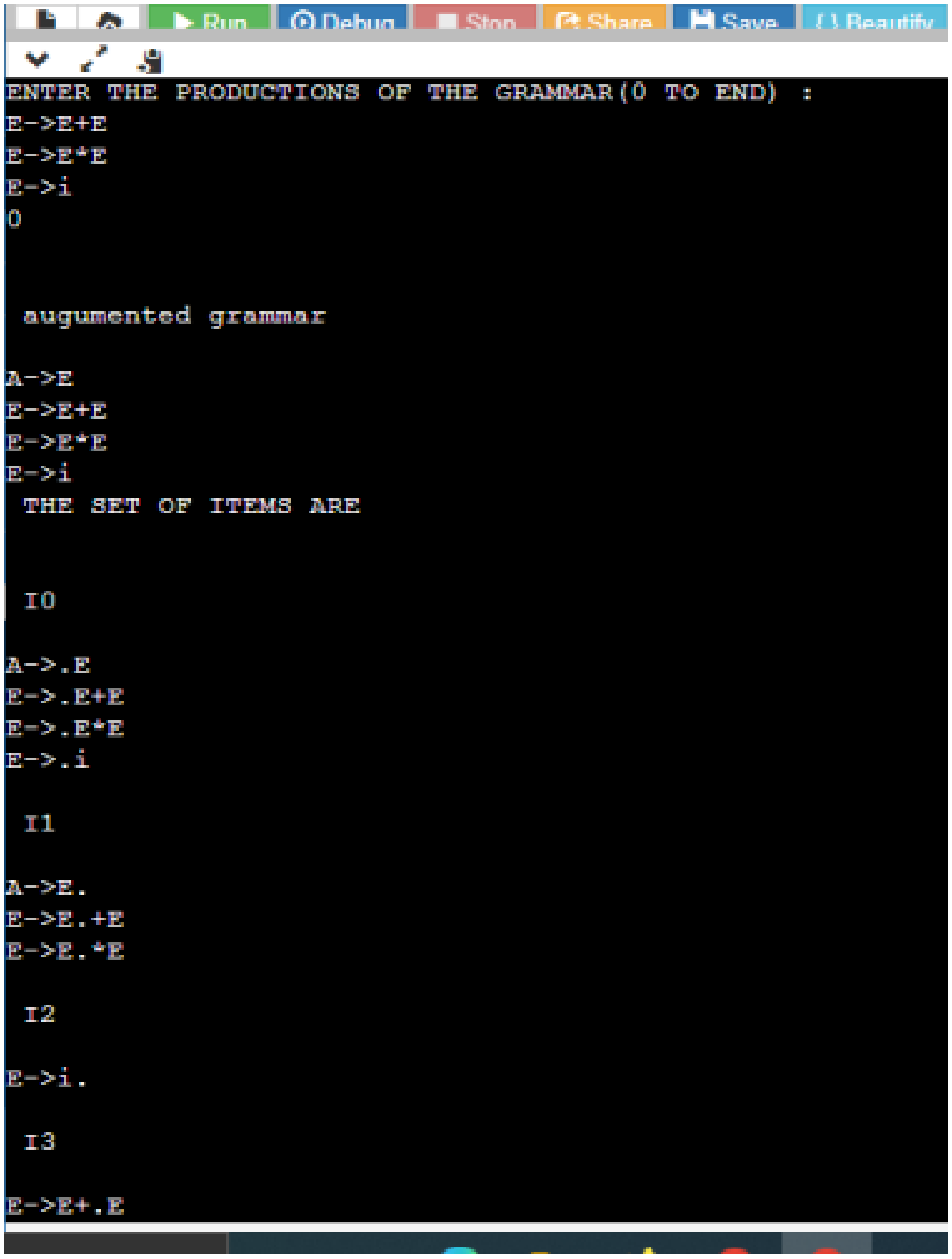
Output Screenshots:

Input:

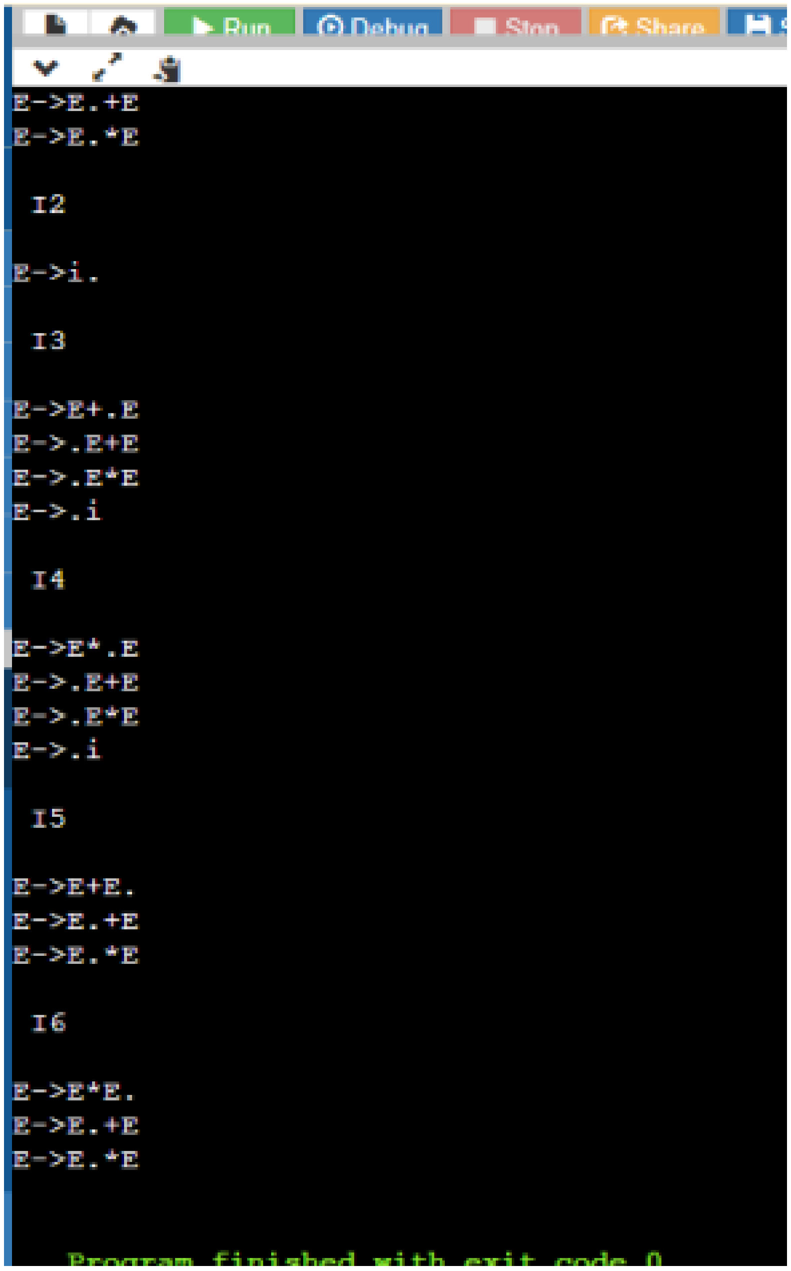
E->E+E

E->E*E

E->id



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


 augumented grammar

A->E
E->E+E
E->E*E
E->i
 THE SET OF ITEMS ARE


 I0

A->.E
E->.E+E
E->.E*E
E->.i

 I1

A->E.
E->E.+E
E->E.*E

 I2

E->i.

 I3

E->E+.E
```



```
E->E.+E
E->E.*E

 I2

E->i.

 I3

E->E+.E
E->.E+E
E->.E*E
E->.i

 I4

E->E*.E
E->.E+E
E->.E*E
E->.i

 I5

E->E+E.
E->E.+E
E->E.*E

 I6

E->E*E.
E->E.+E
E->E.*E


   Program finished with exit code 0
```

Result:

Hence LR(0) Parser was executed and its Implementation was successful

Date: 13 April 2022

Exercise 11

Implementing three address code

Aim:

To study and execute implementation of three address code

Algorithm:

The algorithm takes a sequence of three-address statements as input. For each three address statements of the form a:= b op ¢ 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 ¢ 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:

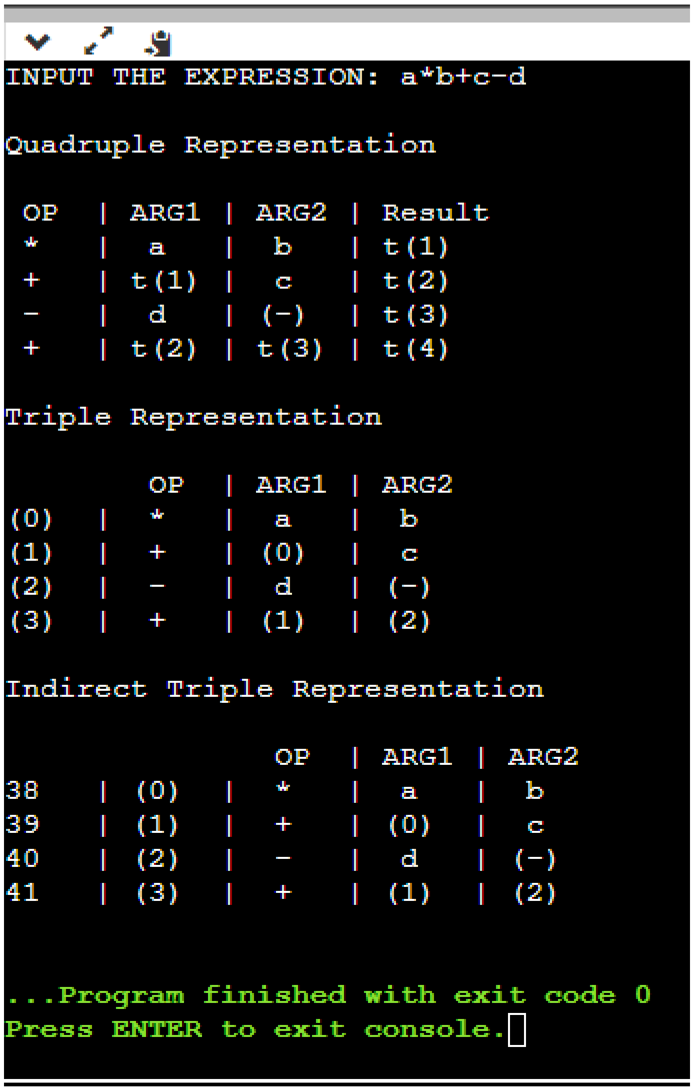
```
import random
OPERATORS = set(['+', '-', '*', '/', '(', ')'])
PRI = {'+':1, '-':1, '*':2, '/':2}

### 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

def Quadruple(pos):
  print('\nQuadruple Representation\n')
  stack = []
  op = []
  x = 1
  print(' OP  | ARG1 | ARG2 | Result')
  for i in pos:
    if i not in OPERATORS:
      stack.append(i)
    elif i == '-':
      op1 = stack.pop()
      stack.append("t(%s)" %x)
      print("{0:^4s} | {1:^4s} | {2:^4s} |{3:4s}".format(i,op1,"(-)"," t(%s)" %x))
      x = x+1
      if stack != []:
        op2 = stack.pop()
        op1 = stack.pop()
        print("{0:^4s} | {1:^4s} | {2:^4s} |{3:4s}".format("+",op1,op2," t(%s)" %x))
        stack.append("t(%s)" %x)
        x = x+1
    elif i == '=':
      op2 = stack.pop()
      op1 = stack.pop()
      print("{0:^4s} | {1:^4s} | {2:^4s} |{3:4s}".format(i,op2,"(-)",op1))
    else:
      op1 = stack.pop()
      op2 = stack.pop()
      print("{0:^4s} | {1:^4s} | {2:^4s} |{3:4s}".format(i,op2,op1," t(%s)" %x))
      stack.append("t(%s)" %x)
      x = x+1

def Triple(pos):
  print('Triple Representation\n')
  print('        OP  | ARG1 | ARG2')
  stack = []
  op = []
  x = 0
  for i in pos:
    if i not in OPERATORS:
      stack.append(i)
    elif i == '-':
      op1 = stack.pop()
      stack.append("(%s)" %x)
      print("{0:4s} | {1:^4s} | {2:^4s} | {3:^4s}".format("(%s)" %x,i,op1,"(-)"))
      x = x+1
      if stack != []:
        op2 = stack.pop()
        op1 = stack.pop()
        print("{0:4s} | {1:^4s} | {2:^4s} | {3:^4s}".format("(%s)" %x,"+",op1,op2))
        stack.append("(%s)" %x)
        x = x+1
    elif i == '=':
      op2 = stack.pop()
      op1 = stack.pop()
      print("{0:4s} | {1:^4s} | {2:^4s} | {3:^4s}".format("(%s)" %x,i,op1,op2))
    else:
      op1 = stack.pop()
      if stack != []:
        op2 = stack.pop()
        print("{0:4s} | {1:^4s} | {2:^4s} | {3:^4s}".format("(%s)" %x,i,op2,op1))
        stack.append("(%s)" %x)
        x = x+1

def intrp(pos):
  print('Indirect Triple Representation\n')
  print('               OP  | ARG1 | ARG2')
  print
  stack = []
  op = []
  x = random.randrange(30,40)
  y = 0
  for i in pos:
    if i not in OPERATORS:
      stack.append(i)
    elif i == '-':
      op1 = stack.pop()
      stack.append("(%s)" %y)
      print("{0:4s} | {1:4s} | {2:^4s} | {3:^4s} | {4:^4s}".format("%s" %x,"(%s)" %y,i,op1,"(-)"))
      x = x+1
      y = y+1
      if stack != []:
        op2 = stack.pop()
        op1 = stack.pop()
        print("{0:4s} | {1:4s} | {2:^4s} | {3:^4s} | {4:^4s}".format("%s" %x,"(%s)"
%y,"+",op1,op2))
        stack.append("(%s)" %y)
        x = x+1
        y = y+1
      elif i == '=':
        op2 = stack.pop()
        op1 = stack.pop()
        print("{0:4s} | {1:4s} | {2:^4s} | {3:^4s} | {4:^4s}".format("%s" %x,"(%s)" %y,i,op1,op2))
    else:
      op1 = stack.pop()
      if stack != []:
        op2 = stack.pop()
        print("{0:4s} | {1:4s} | {2:^4s} | {3:^4s} | {4:^4s}".format("%s" %x,"(%s)" %y,i,op2,op1))
        stack.append("(%s)" %y)
        x = x+1
        y = y+1

expres = input("INPUT THE EXPRESSION: ")
pos = infix_to_postfix(expres)
Quadruple(pos)
print()
Triple(pos)
print()
intrp(pos)
```

Output Screenshots:

Input:

a*b+c-d

Output:



```
INPUT THE EXPRESSION: a*b+c-d

Quadruple Representation

 OP   | ARG1 | ARG2 | Result
 *    |  a   |  b   | t(1)
 +    | t(1) |  c   | t(2)
 -    |  d   | (-)  | t(3)
 +    | t(2) | t(3) | t(4)

Triple Representation

       OP   | ARG1 | ARG2
(0)  |  *   |  a   |  b
(1)  |  +   | (0)  |  c
(2)  |  -   |  d   | (-)
(3)  |  +   | (1)  | (2)

Indirect Triple Representation

              OP   | ARG1 | ARG2
38   | (0)  |  *   |  a   |  b
39   | (1)  |  +   | (0)  |  c
40   | (2)  |  -   |  d   | (-)
41   | (3)  |  +   | (1)  | (2)


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

Result:

Three address code Implementation was successful

Date:20th April 2022
CLA P4 Assessment

Implementing Operator Precedence

Aim:
To write a program to implement and execute operator precedence.
Program :

```
#include <bits/stdc++.h>
using namespace std;
string wordsk[20];
int wordi=0;

bool isNumber(string s)
{
    for (int i = 0; i < s.length(); i++)
        if (isdigit(s[i]) == false)
            return false;

    return true;
}

void goodstring(string str)
{
   string word = "";
   for (auto x : str)
   {
       if (x == ' ')
       {
           wordsk[wordi]=word;
           wordi++;
           word = "";
       }
       else
       {
           word = word + x;
       }
   }
wordsk[wordi]=word;
wordi++;
}

bool a=false,b=false,c=false,d=false,e=false;
```

```
int precedence(char op){
        if(op == '+'||op == '-')
        return 1;
        if(op == '*'||op == '/')
        return 2;
        if (op == '^')
   return 3;

        return 0;
}

float applyOp(float a, float b, char op){

        switch(op){
                case '+': return a + b;
                case '-': return a - b;
                case '*': return a * b;
                case '/': return a / b;

                case '^': return pow(a,b);
        }
}


float evaluate(string tokens){
        int i;

        // stack to store integer values.
        stack <float> values;

        // stack to store operators.
        stack <char> ops;

        for(i = 0; i < tokens.length(); i++){

                // Current token is a whitespace,
                // skip it.
                if(tokens[i] == ' ')
                        continue;

                else if(isdigit(tokens[i])){
                        int val = 0;

                        // There may be more than one
                        // digits in number.
                        while(i < tokens.length() &&
                                    isdigit(tokens[i]))
                        {
                            val = (val*10) + (tokens[i]-'0');
                            i++;
                        }

                        values.push(val);
                }

                // Current token is an operator.
                else
                {
                        // While top of 'ops' has same or greater
                        // precedence to current token, which
                        // is an operator. Apply operator on top
                        // of 'ops' to top two elements in values stack.
                        while(!ops.empty() && precedence(ops.top())
                                            >= precedence(tokens[i])){
                            float val2 = values.top();
                            values.pop();

                            float val1 = values.top();
                            values.pop();

                            char op = ops.top();
                            ops.pop();

                            values.push(applyOp(val1, val2, op));

                        }

                        // Push current token to 'ops'.
                        ops.push(tokens[i]);
                }
        }

        // Entire expression has been parsed at this
        // point, apply remaining ops to remaining
        // values.
        while(!ops.empty()){
                float val2 = values.top();
                values.pop();
```

```
                float val1 = values.top();
                values.pop();

                char op = ops.top();
                ops.pop();

                values.push(applyOp(val1, val2, op));
        }

        // Top of 'values' contains result, return it.
        return values.top();
}

void valid(string wordsk[20])
{
   int cnt=0;
   while (wordsk[cnt]!= "\0")
   {
      cnt++;
   }

   if (!isNumber(wordsk[cnt-1]))
   {
      cout<<"wrong string"<<endl;
      exit(100);
   }
   for (int k=1;k<cnt;k+=2)
   {
      if (wordsk[k]=="+" || wordsk[k]=="-" || wordsk[k]=="*" || wordsk[k]=="/" || wordsk [k]=="^")
      {
         continue;
      }
      else
      {
         cout<<"wrong string"<<endl;
         exit(100);
      }
   }
   for (int k=0;k<cnt;k+=2)
   {
      if (!isNumber(wordsk[k]))
      {
         cout<<"error";
```

```
            exit(100);
        }
    }

}

void printing(char ar[6],int i,int k)
{
    cout<<"\t";
    cout << ar[k];
        for (int l=1;l<i;l++)
        {
            if (precedence(ar[k])>precedence(ar[l]))
            {
                cout <<" "<< '>' ;
            }
            else if (precedence(ar[k])==precedence(ar[l]) && precedence(ar[l])!=3)
            {
                cout<<" "<< '>';
            }
            else if (precedence(ar[k])< precedence(ar[l]))
            {
                cout<<" "<<'<';
            }
            else if (precedence(ar[k])==precedence(ar[l]) && precedence(ar[l])==3)
            {
                cout<<" "<<'<';
            }
        }
        cout << endl;
}

int main()
{
    cout << "enter a expression \nin a manner that the operators and operands are seperated by
a single space \nand only ^,*,/,+,- are allowed"<< "\n";
    string tokens;
    getline(cin,tokens);

    goodstring(tokens);
    valid(wordsk);

    cout<< endl;
    for (int i=0;i<tokens.length();i++)
```

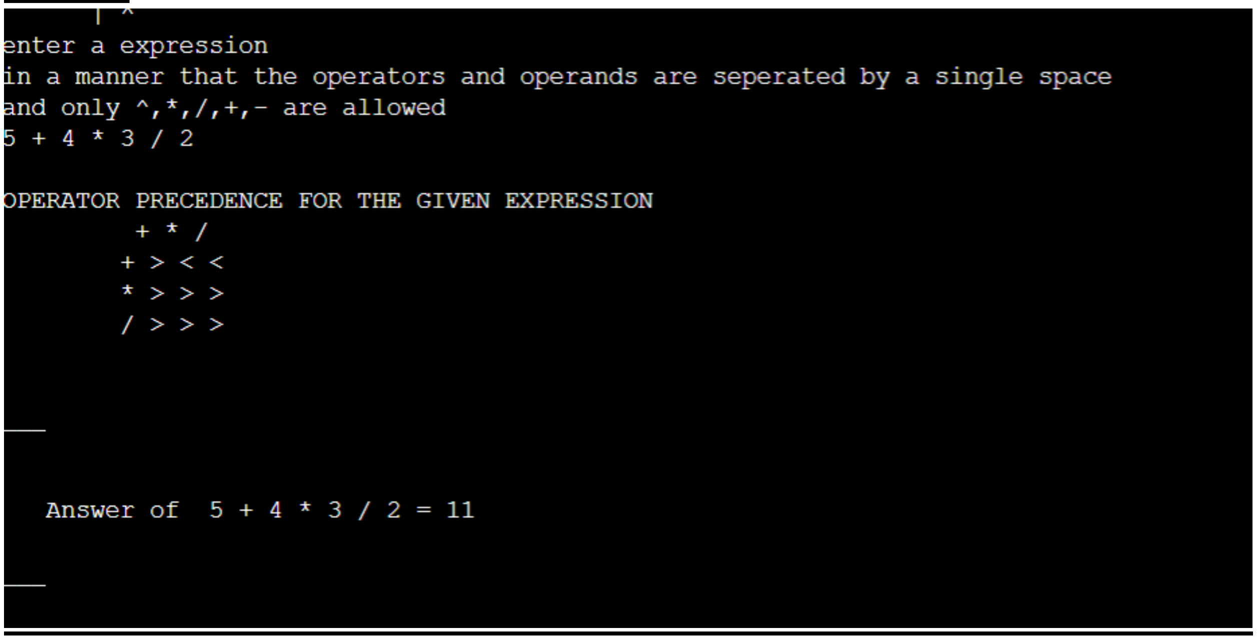
```
{
    switch (tokens[i])
    {
        case '+':a=true;
        break;
        case '-':b=true;
        break;
        case '*':c=true;
        break;
        case '/':d=true;
        break;
        case '^':e=true;
        break;
    }
}
//cout<< a <<" "<< b << " " << c << " " << d << endl;

char ar[6];
int i=1;

if (a==true)
{
    ar[i]='+';
    i++;
}
if (b==true)
{
    ar[i]='-';
    i++;
}
if (c==true)
{
    ar[i]='*';
    i++;
}
if (d==true)
{
    ar[i]='/';
    i++;
}
if (e==true)
{
    ar[i]='^';
    i++;
    }
cout<<"OPERATOR PRECEDENCE FOR THE GIVEN EXPRESSION"<<endl;
cout<<"\t";
for (int j=0;j<i;j++)
    cout << ar[j]<< " ";
    cout<<endl;

for (int k=1;k<i;k++)
    {
        printing(ar,i,k);
    }
cout<<endl<<endl;
cout<<"___"<<endl<<endl<<endl;

    cout<<"   "<<"Answer of  "<<tokens;
    cout<<" = "<< evaluate(tokens)<< endl;
    cout<<endl;
cout<<"___"<<endl<<endl<<endl;
    return 0;}
```

Output Screenshots:

Input:

5+4*3/2

Output:



```
enter a expression
in a manner that the operators and operands are seperated by a single space
and only ^,*,/,+,- are allowed
5 + 4 * 3 / 2

OPERATOR PRECEDENCE FOR THE GIVEN EXPRESSION
          + * /
        + > < <
        * > > >
        / > > >



___


   Answer of  5 + 4 * 3 / 2 = 11


___
```

Result:

Operator Precedence program was hence executed and implemented.

Date: 20th April 2022
CLA P4 Assessment

Generate the object code for the given input

Aim:
To write a program to generate the object code for the given input

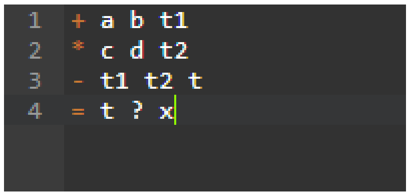
Algorithm:

Program:
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char op[2],arg1[5],arg2[5],result[5];
void main()
{
  FILE *fp1,*fp2;
  fp1=fopen("input.txt","r");
  fp2=fopen("output.txt","w");
  while(!feof(fp1))
  {

    fscanf(fp1,"%s%s%s%s",op,arg1,arg2,result);
    if(strcmp(op,"+")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
      fprintf(fp2,"\nADD R0,%s",arg2);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
     if(strcmp(op,"*")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
      fprintf(fp2,"\nMUL R0,%s",arg2);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
    if(strcmp(op,"-")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
      fprintf(fp2,"\nSUB R0,%s",arg2);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
      if(strcmp(op,"/")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
      fprintf(fp2,"\nDIV R0,%s",arg2);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
if(strcmp(op,"=")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
    fprintf(fp2,"\nMOV %s,R0",result);
  }
  }
  fclose(fp1);
  fclose(fp2);
  getch();
}
```

Output Screenshots :

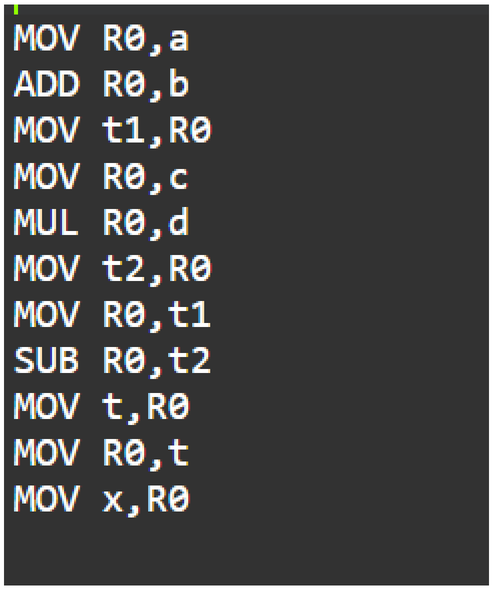
Input:

+ a b t1
* c d t2
- t1 t2 t
= t ? x



```
+ a b t1
* c d t2
- t1 t2 t
= t ? x
```

Output:



```
MOV R0,a
ADD R0,b
MOV t1,R0
MOV R0,c
MUL R0,d
MOV t2,R0
MOV R0,t1
SUB R0,t2
MOV t,R0
MOV R0,t
MOV x,R0
```

Result:

Successfully generated the object code for the given input.