

INDEX

S No.	Title	Date
1.	Implementation of Lexical Analyzer	Jan 10
2.	Conversion from Regular Expression to NFA	Jan 27
3.	Conversion from NFA to DFA	Feb 8
4.	Elimination of Left Recursion and Left Factoring	Feb 17
5.	Computation of FIRST AND FOLLOW	Feb 22
6.	Construction of Predictive Parsing Table	March 4
7.	Implementation of Shift Reduce Parsing	March 10
8.	Computation of LEADING AND TRAILING	March 17
9..	Computation of LR (0) items	March 24
10.	Intermediate code generation – Postfix, Prefix	March 31
11.	Quadruple triple	March 31
12.	Hackerrank Regex Programs	March 31

LEXICAL ANALYZER

EX. NO. 1

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

ALGORITHM:

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

CODE:

week_1.cpp CD LABS

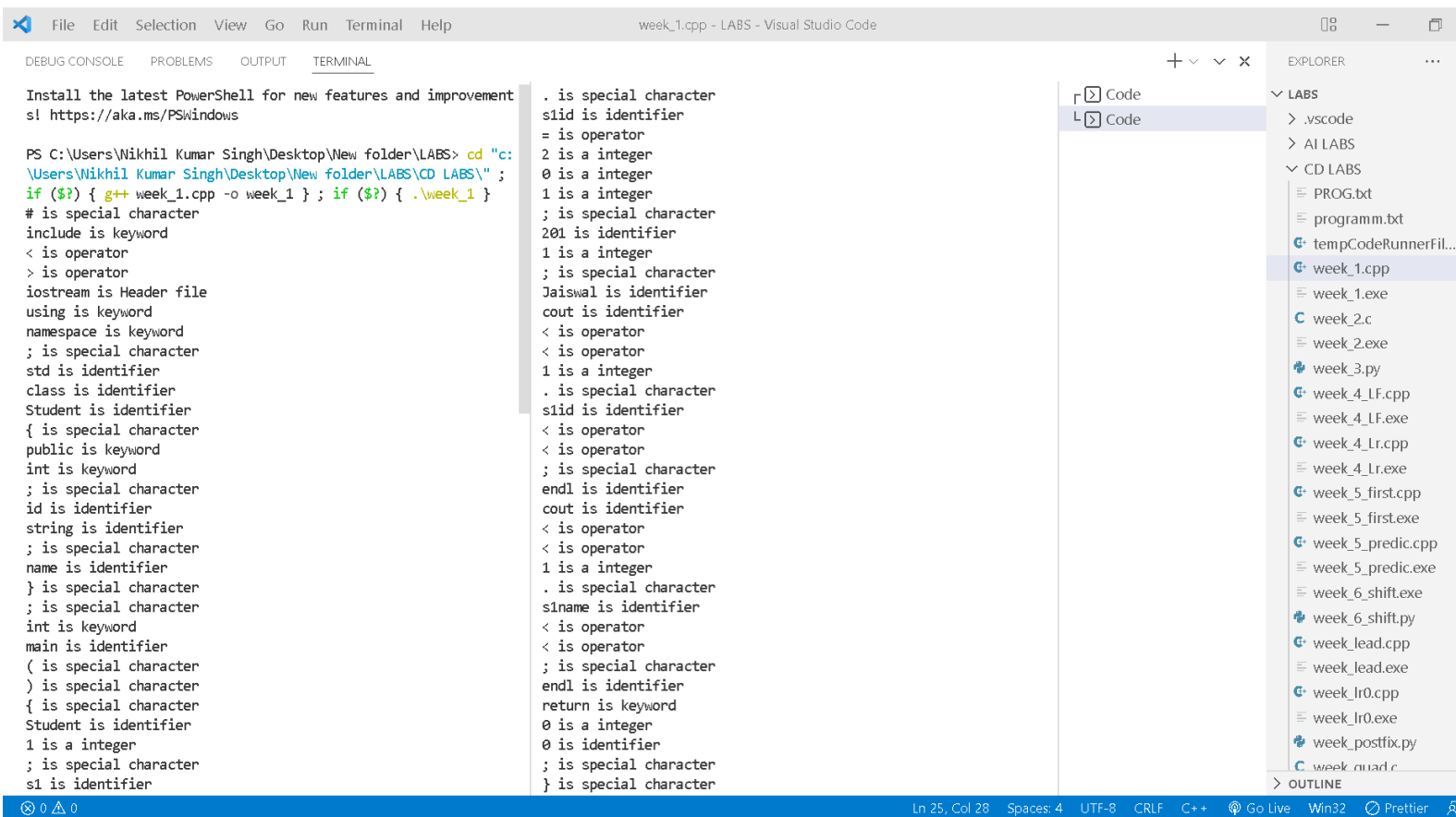
week_1.cpp

week_1.cpp

```
// lexical analyser
//RA1911033010102 - Nikhil Kumar Singh
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include<vector>
int isKeyword(char buffer[]){
char keywords[36][10] =
{"auto", "using", "namespace", "include", "break", "case", "char", "const", "continue",
"default", "do", "double", "else", "enum", "extern", "float", "for", "goto",
"if", "int", "long", "public", "register", "return", "short", "signed",
"sizeof", "static", "struct", "switch", "typedef", "union",
"unsigned", "void", "volatile", "while"};
int i, flag = 0;
for(i = 0; i < 32; ++i){
if(strcmp(keywords[i], buffer) == 0){
flag = 1;
break;}}
return flag;
}
char isHeader(char buffer[]){
char headers[2][10] = {
{"iostream", "stdio"}
};
int i, flag = 0;
for(i = 0; i < 2; ++i){
if(strcmp(headers[i], buffer) == 0){
flag = 1;
break;}}
return flag;
}
```

```
int main(){
char ch, buffer[15], operators[] = "+-*/%=<>", special[] = "#;,.{}[]()";
FILE *fp;
int i,j=0;
fp = fopen("PROG.txt", "r");
if(fp == NULL){
printf("error while opening the file\n");
exit(0);}
while((ch = fgetc(fp)) != EOF){
for(i = 0; i < 8; ++i){
if(ch == operators[i])
printf("%c is operator\n", ch);
}
for(i = 0; i < 10; ++i){
if(ch == special[i])
printf("%c is special character\n", ch);
}
if(isdigit(ch)==true)
printf("%c is a integer\n",ch);
if(isalnum(ch)){
buffer[j++] = ch;
}
else if((ch == ' ' || ch == '\n') && (j != 0)){
buffer[j] = '\0';
j = 0;
if(isKeyword(buffer) == 1)
printf("%s is keyword\n", buffer);
else if(isHeader(buffer)==1)
printf("%s is Header file\n", buffer);
else
printf("%s is identifier\n", buffer);
}
}
fclose(fp);
return 0;
}
```

OUTPUT:



```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS> cd "c:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS\CD LABS\" ; if ($?) { g++ week_1.cpp -o week_1 } ; if ($?) { .\week_1 }
# is special character
include is keyword
< is operator
> is operator
iostream is Header file
using is keyword
namespace is keyword
; is special character
std is identifier
class is identifier
Student is identifier
{ is special character
public is keyword
int is keyword
; is special character
id is identifier
string is identifier
; is special character
name is identifier
} is special character
; is special character
int is keyword
main is identifier
( is special character
) is special character
{ is special character
Student is identifier
1 is a integer
; is special character
s1 is identifier
. is special character
s1id is identifier
= is operator
2 is a integer
0 is a integer
1 is a integer
; is special character
201 is identifier
1 is a integer
; is special character
Jaishwal is identifier
cout is identifier
< is operator
< is operator
1 is a integer
. is special character
s1id is identifier
< is operator
< is operator
; is special character
endl is identifier
cout is identifier
< is operator
< is operator
1 is a integer
. is special character
s1name is identifier
< is operator
< is operator
; is special character
endl is identifier
return is keyword
0 is a integer
0 is identifier
; is special character
} is special character
```

EXPLORER

- LABS
 - .vscode
 - AI LABS
 - CD LABS
 - PROG.txt
 - programm.txt
 - tempCodeRunnerFil...
 - week_1.cpp
 - week_1.exe
 - week_2.c
 - week_2.exe
 - week_3.py
 - week_4_LF.cpp
 - week_4_LF.exe
 - week_4_Lr.cpp
 - week_4_Lr.exe
 - week_5_first.cpp
 - week_5_first.exe
 - week_5_predic.cpp
 - week_5_predic.exe
 - week_6_shift.exe
 - week_6_shift.py
 - week_lead.cpp
 - week_lead.exe
 - week_lr0.cpp
 - week_lr0.exe
 - week_postfix.py
 - week_quad.c

OUTLINE

Ln 25, Col 28 Spaces: 4 UTF-8 CRLF C++ Go Live Win32 Prettier

RESULT :

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

EX. NO. 2

RE to NFA

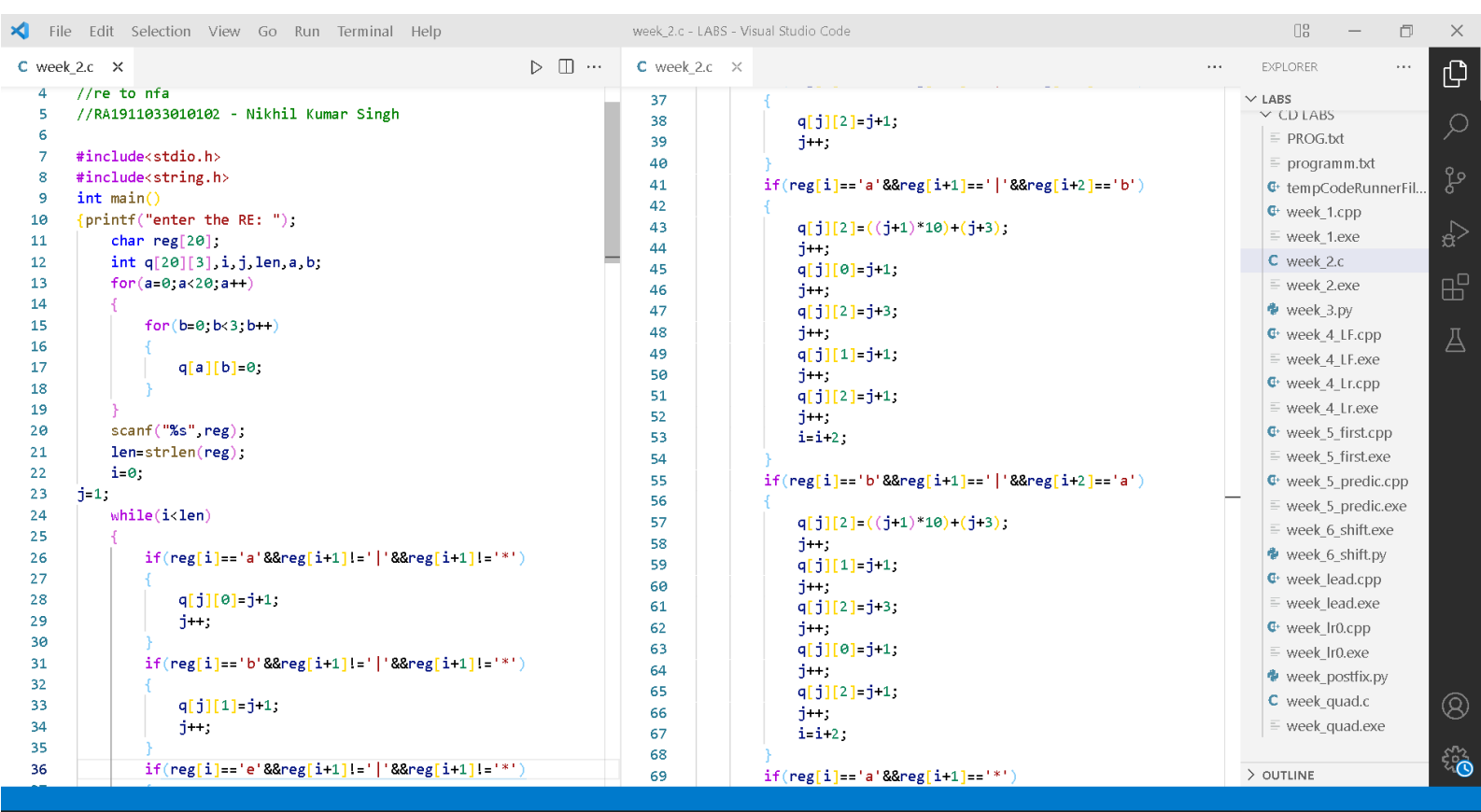
AIM: To convert Regular Expression to NFA

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:

CODE:



```
4 //re to nfa
5 //RA1911033010102 - Nikhil Kumar Singh
6
7 #include<stdio.h>
8 #include<string.h>
9 int main()
10 {printf("enter the RE: ");
11   char reg[20];
12   int q[20][3],i,j,len,a,b;
13   for(a=0;a<20;a++)
14   {
15     for(b=0;b<3;b++)
16     {
17       q[a][b]=0;
18     }
19   }
20   scanf("%s",reg);
21   len=strlen(reg);
22   i=0;
23   j=1;
24   while(i<len)
25   {
26     if(reg[i]=='a'&&reg[i+1]!='|'&&reg[i+1]!='*')
27     {
28       q[j][0]=j+1;
29       j++;
30     }
31     if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='*')
32     {
33       q[j][1]=j+1;
34       j++;
35     }
36     if(reg[i]=='e'&&reg[i+1]!='|'&&reg[i+1]!='*')
37
38
39
40
41     if(reg[i]=='a'&&reg[i+1]=='|'&&reg[i+2]=='b')
42     {
43       q[j][2]=(j+1)*10+(j+3);
44       j++;
45       q[j][0]=j+1;
46       j++;
47       q[j][2]=j+3;
48       j++;
49       q[j][1]=j+1;
50       j++;
51       q[j][2]=j+1;
52       j++;
53       i=i+2;
54     }
55     if(reg[i]=='b'&&reg[i+1]=='|'&&reg[i+2]=='a')
56     {
57       q[j][2]=(j+1)*10+(j+3);
58       j++;
59       q[j][1]=j+1;
60       j++;
61       q[j][2]=j+3;
62       j++;
63       q[j][0]=j+1;
64       j++;
65       q[j][2]=j+1;
66       j++;
67       i=i+2;
68     }
69     if(reg[i]=='a'&&reg[i+1]=='*')
```

```
File Edit Selection View Go Run Terminal Help week_2.c - LABS - Visual Studio Code
C week_2.c x ... C week_2.c x
69 if(reg[i]=='a' && reg[i+1]!='*')
70 {
71     q[j][2] = ((j+1)*10) + (j+3);
72     j++;
73     q[j][0] = j+1;
74     j++;
75     q[j][2] = ((j+1)*10) + (j-1);
76     j++;
77 }
78 if(reg[i]=='b' && reg[i+1]!='*')
79 {
80     q[j][2] = ((j+1)*10) + (j+3);
81     j++;
82     q[j][1] = j+1;
83     j++;
84     q[j][2] = ((j+1)*10) + (j-1);
85     j++;
86 }
87 if(reg[i]==' ' && reg[i+1]!='*')
88 {
89     q[0][2] = ((j+1)*10) + 1;
90     q[j][2] = ((j+1)*10) + 1;
91     j++;
92 }
93 i++;
94 }
95 printf("Transition function \n");
96 for(i=0; i<=j; i++)
97 {
98     if(q[i][0]!=0)
99         printf("\n q[%d,a]--->%d", i, q[i][0]);
100     if(q[i][1]!=0)
101         printf("\n q[%d,b]--->%d", i, q[i][1]);
102     if(q[i][2]!=0)
103     {
104         if(q[i][2]<10)
105             printf("\n q[%d,e]--->%d", i, q[i][2]);
106         else
107             printf("\n q[%d,e]--->%d & %d", i, q[i][2]/10, q[i][2]%10);
108     }
109 }
110 return 0;
111 }
```

OUTPUT :

```
File Edit Selection View Go Run Terminal Help week_2.c - LABS - Visual Studio Code
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL
ek_2 }; if ($?) { .\week_2 }
enter the RE: a*b
Transition function
q[1,e]--->2 & 4
q[2,a]--->3
q[3,e]--->4 & 2
q[4,b]--->5
PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS\CD LABS>
```

RESULT :

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

EX. NO. 3

AIM: To convert 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 $S = e\text{-Closure}(\text{MoveNFA}(T,a))$
 - c. if S is not in SDFA already then, add S to SDFA (as an “unmarked” state)
 - d. Set $\text{MoveDFA}(T,a)$ to S
5. For each S in SDFA if any s & S is a final state in the NFA then, mark S as a final state in the DFA
6. Print the result.
7. Stop the program

PROGRAM:

CODE:

```
File Edit Selection View Go Run Terminal Help week_3.py - LABS - Visual Studio Code
week_3.py X week_3.py X
3 #nfa to dfa
4 #RA1911033010102 - Nikhil Kumar Singh
5 import pandas as pd
6 nfa = {}
7 n = int(input("No. of states : "))
8 t = int(input("No. of transitions : "))
9 for i in range(n):
10     state = input("state name : ")
11     nfa[state] = {}
12     for j in range(t):
13         path = input("path : ")
14         print("Enter end state from state {} travelling through path {} : ".format(state, path))
15         reaching_state = [x for x in input().split()]
16         nfa[state][path] = reaching_state
17
18 print("\nNFA :- \n")
19 print(nfa)
20 print("\nPrinting NFA table :- ")
21 nfa_table = pd.DataFrame(nfa)
22 print(nfa_table.transpose())
23
24 print("Enter final state of NFA : ")
25 nfa_final_state = [x for x in input().split()]
26 new_states_list = []
27 dfa = {}
28 keys_list = list(
29     list(nfa.keys())[0])
30 path_list = list(nfa[keys_list[0]].keys())
31 dfa[keys_list[0]] = {}
32 for y in range(t):
33     var = ""
34     for i in range(len(nfa[keys_list[0]][path_list[y]])):
35         dfa[keys_list[0]][path_list[y]] = var
36
37 new_states_list.append(var)
38 keys_list.append(var)
39 while len(new_states_list) != 0:
40     dfa[new_states_list[0]] = {}
41     for _ in range(len(new_states_list[0])):
42         for i in range(len(path_list)):
43             temp = []
44             for j in range(len(new_states_list[0])):
45                 temp += nfa[new_states_list[0][j]][path_list[i]]
46             s = ""
47             s = s.join(temp)
48             if s not in keys_list:
49                 new_states_list.append(s)
50                 keys_list.append(s)
51             dfa[new_states_list[0]][path_list[i]] = s
52
53 new_states_list.remove(new_states_list[0])
54 print("\nDFA :- \n")
55 print(dfa)
56 print("\n DFA table :- ")
57 dfa_table = pd.DataFrame(dfa)
58 print(dfa_table.transpose())
59
60 dfa_states_list = list(dfa.keys())
61 dfa_final_states = []
62 for x in dfa_states_list:
63     for i in x:
64         if i in nfa_final_state:
65             dfa_final_states.append(x)
66             break
67
68 print("\nFinal states of the DFA are : ", dfa_final_states)
```

OUTPUT:

File Edit Selection View Go Run Terminal Help week_3.py - LABS - Visual Studio Code

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

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

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

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

Printing NFA table :-

	0	1
A	[A]	[AB]
B	[C]	[C]
C	[]	[]

Enter final state of NFA :
C

DFA :-

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

DFA table :-

	0	1
A	A	AB
AB	AC	ABC
AC	A	AB
ABC	AC	ABC

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

RESULT :

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

ELIMINATION OF LEFT RECURSION

EX. NO. 4(a)

AIM: A program for Elimination of Left Recursion.

ALGORITHM:

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

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_m$

$A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$

Then replace it by

$A \rightarrow \beta_i A' \quad i=1,2,3,\dots,m$

$A' \rightarrow \alpha_j A' \quad j=1,2,3,\dots,n$

$A' \rightarrow \epsilon$

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

7. Stop.

CODE :

```
File Edit Selection View Go Run Terminal Help
week_4_Lr.cpp - LABS - Visual Studio Code

week_4_Lr.cpp
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  using namespace std;
5  int main()
6  {int n;
7   cout<<"\n number of non terminals ";
8   cin>>n;
9   cout<<"\n Enter non terminals : ";
10  int i;
11  vector<string> nonter(n);
12  vector<int> leftrecr(n,0);
13  for(i=0;i<n;++i) {
14   cout<<"\n Non terminal "<<i+1<<" : ";
15   cin>>nonter[i];
16   vector<vector<string>> prod;
17   cout<<"\nEnter '^' for null";
18   for(i=0;i<n;++i) {
19    cout<<"\nNumber of "<<nonter[i]<<" productions: ";
20    int k;
21    cin>>k;
22    int j;
23    cout<<"\nOn enter all "<<nonter[i]<<" productions";
24    vector<string> temp(k);
25    for(j=0;j<k;++j) {
26     cout<<"\nRHS of production "<<j+1<<" : ";
27     string abc;
28     cin>>abc;
29     temp[j]=abc;
30     if(nonter[i].length()<=abc.length()&&nonter[i].compare(abc.substr
31     leftrecr[i]=1;
32     prod.push_back(temp);
33     for(i=0;i<n;++i) {
34      prod.push_back(temp);
35      for(i=0;i<n;++i) {
36       if(leftrecr[i]==0)
37        continue;
38       int j;
39       nonter.push_back(nonter[i]+""");
40       vector<string> temp;
41       for(j=0;j<prod[i].size();++j) {
42        if(nonter[i].length()<=prod[i][j].length()&&nonter[i].compare(pr
43        ()))==0)
44         {string abc=prod[i][j].substr(nonter[i].length(),prod[i][j].le
45         temp.push_back(abc);
46         prod[i].erase(prod[i].begin()+j);
47         --j; }
48       else {
49        prod[i][j]+=nonter[i]+"""; } }
50       temp.push_back("");
51       prod.push_back(temp);
52       cout<<"\n\n";
53       cout<<"\nNew non-terminals: ";
54       for(i=0;i<nonter.size();++i)
55        cout<<nonter[i]<<" ";
56       cout<<"\n\nNew productions: ";
57       for(i=0;i<nonter.size();++i) {
58        int j;
59        for(j=0;j<prod[i].size();++j) {
60         cout<<"\n"<<nonter[i]<<" -> "<<prod[i][j];}}
61       return 0;
62     }
```


OUTPUT:

```

File Edit Selection View Go Run Terminal Help
week_4_Lr.cpp - LABS - Visual Studio C

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

number of non terminals 3

Enter non terminals :
Non terminal 1 : E

Non terminal 2 : T

Non terminal 3 : F

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

On enter all E productions
RHS of production 1: E+T

RHS of production 2: T

Number of T productions: 2

On enter all T productions
RHS of production 1: T*F

RHS of production 2: F

Number of F productions: 2

On enter all F productions
RHS of production 1: (E)

RHS of production 2: i
110

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

New productions:
E -> TE'
T -> FT'
F -> (E)
F -> i
E' -> +TE'
E' -> ^
T' -> *FT'
T' -> ^
PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS\CD LABS> 
```

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

ELIMINATION OF LEFT RECURSION

EX. NO. 4(a)

AIM: A program for Elimination of Left Recursion.

ALGORITHM:

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

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_m$

$A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$

Then replace it by

$A \rightarrow \beta_i A' \quad i=1,2,3,\dots,m$

$A' \rightarrow \alpha_j A' \quad j=1,2,3,\dots,n$

$A' \rightarrow \epsilon$

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

7. Stop.

CODE :

```
File Edit Selection View Go Run Terminal Help
week_4_Lr.cpp - LABS - Visual Studio Code

week_4_Lr.cpp
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  using namespace std;
5  int main()
6  {int n;
7   cout<<"\n number of non terminals ";
8   cin>>n;
9   cout<<"\n Enter non terminals : ";
10  int i;
11  vector<string> nonter(n);
12  vector<int> leftrecr(n,0);
13  for(i=0;i<n;++i) {
14   cout<<"\n Non terminal "<<i+1<<" : ";
15   cin>>nonter[i];
16   vector<vector<string>> prod;
17   cout<<"\nEnter '^' for null";
18   for(i=0;i<n;++i) {
19    cout<<"\nNumber of "<<nonter[i]<<" productions: ";
20    int k;
21    cin>>k;
22    int j;
23    cout<<"\nOn enter all "<<nonter[i]<<" productions";
24    vector<string> temp(k);
25    for(j=0;j<k;++j) {
26     cout<<"\nRHS of production "<<j+1<<" : ";
27     string abc;
28     cin>>abc;
29     temp[j]=abc;
30     if(nonter[i].length()<=abc.length()&&nonter[i].compare(abc.substr
31     leftrecr[i]=1;
32     prod.push_back(temp);
33     for(i=0;i<n;++i) {
34      prod.push_back(temp);
35      for(i=0;i<n;++i) {
36       if(leftrecr[i]==0)
37        continue;
38       int j;
39       nonter.push_back(nonter[i]+""");
40       vector<string> temp;
41       for(j=0;j<prod[i].size();++j) {
42        if(nonter[i].length()<=prod[i][j].length()&&nonter[i].compare(pr
43        ))==0)
44         {string abc=prod[i][j].substr(nonter[i].length(),prod[i][j].le
45         temp.push_back(abc);
46         prod[i].erase(prod[i].begin()+j);
47         --j; }
48        else {
49         prod[i][j]+=nonter[i]+"""; } }
50        temp.push_back("");
51        prod.push_back(temp);
52        cout<<"\n\n";
53        cout<<"\nNew non-terminals: ";
54        for(i=0;i<nonter.size();++i)
55         cout<<nonter[i]<<" ";
56        cout<<"\n\nNew productions: ";
57        for(i=0;i<nonter.size();++i) {
58         int j;
59         for(j=0;j<prod[i].size();++j) {
60          cout<<"\n"<<nonter[i]<<" -> "<<prod[i][j]; }
61        return 0;
62      }
```

OUTPUT:

```

File Edit Selection View Go Run Terminal Help
week_4_Lr.cpp - LABS - Visual Studio C

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

number of non terminals 3

Enter non terminals :
Non terminal 1 : E

Non terminal 2 : T

Non terminal 3 : F

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

On enter all E productions
RHS of production 1: E+T

RHS of production 2: T

Number of T productions: 2

On enter all T productions
RHS of production 1: T*F

RHS of production 2: F

Number of F productions: 2

On enter all F productions
RHS of production 1: (E)

RHS of production 2: i
110

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

New productions:
E -> TE'
T -> FT'
F -> (E)
F -> i
E' -> +TE'
E' -> ^
T' -> *FT'
T' -> ^
PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS\CD LABS> 
```

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

FIRST AND FOLLOW

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

ALGORITHM:

For computing the first:

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

Example: $F \rightarrow I \mid id$

We can write it as $\text{FIRST}(F) \rightarrow \{ (, id)$

2. If X is a non-terminal like $E \rightarrow T$ then to get $\text{FIRST}(E)$ substitute T with other productions

until you get a terminal as the first symbol

3. If $X \rightarrow \epsilon$ then add ϵ to $\text{FIRST}(X)$.

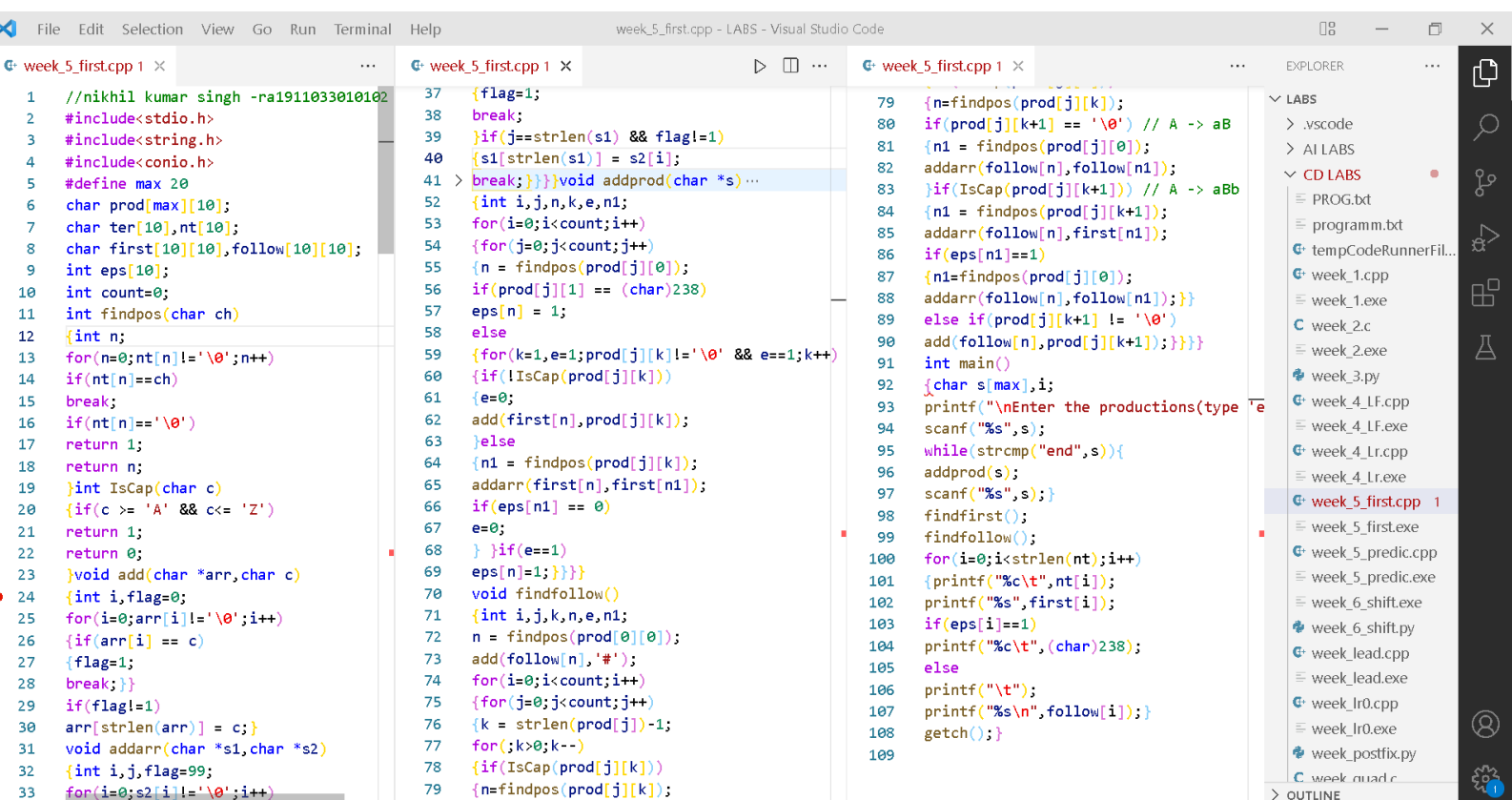
For computing the follow:

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

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

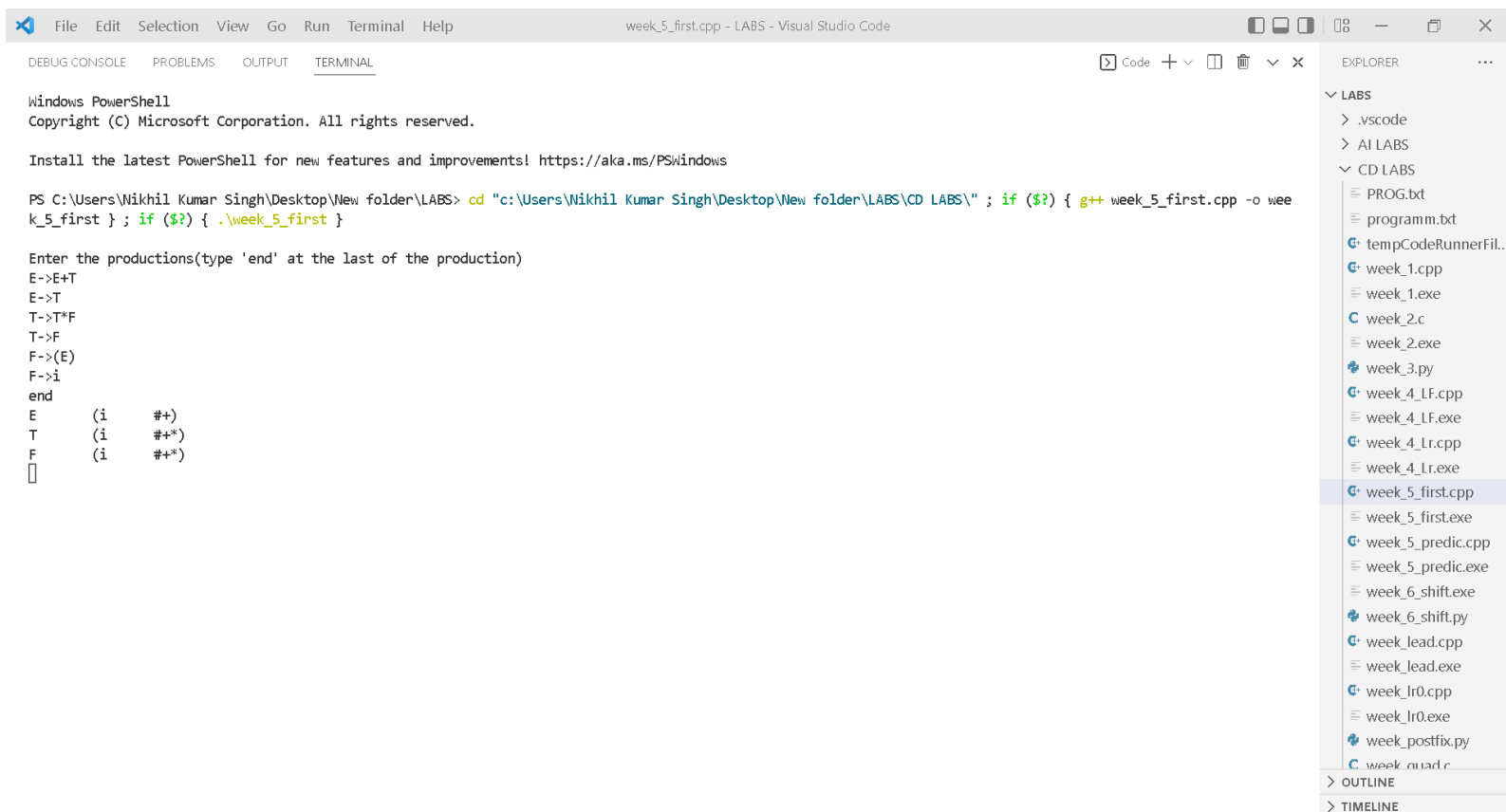
(b) If that non-terminal is followed by any other non-terminal then add FIRST of other nonterminal into the FOLLOW set.

CODE :



```
1 //nikhil kumar singh -ra1911033010102
2 #include<stdio.h>
3 #include<string.h>
4 #include<conio.h>
5 #define max 20
6 char prod[max][10];
7 char ter[10],nt[10];
8 char first[10][10],follow[10][10];
9 int eps[10];
10 int count=0;
11 int findpos(char ch)
12 {int n;
13 for(n=0;nt[n]!='\0';n++)
14 if(nt[n]==ch)
15 break;
16 if(nt[n]=='\0')
17 return 1;
18 return n;
19 }int IsCap(char c)
20 {if(c >= 'A' && c <= 'Z')
21 return 1;
22 return 0;
23 }void add(char *arr,char c)
24 {int i,flag=0;
25 for(i=0;arr[i]!='\0';i++)
26 {if(arr[i] == c)
27 {flag=1;
28 break;}}
29 if(flag!=1)
30 arr[strlen(arr)] = c;
31 void addarr(char *s1,char *s2)
32 {int i,j,flag=0;
33 for(i=0;s2[i]!='\0';i++)
34 {flag=1;
35 break;}}
36 void addprod(char *s)
37 {flag=1;
38 break;
39 }if(j==strlen(s1) && flag!=1)
40 {s1[strlen(s1)] = s2[i];
41 > break;}}void addprod(char *s)
42 {int i,j,n,k,e,n1;
43 for(i=0;i<count;i++)
44 {for(j=0;j<count;j++)
45 {n = findpos(prod[j][0]);
46 if(prod[j][1] == (char)238)
47 eps[n] = 1;
48 else
49 {for(k=1,e=1;prod[j][k]!='\0' && e==1;k++)
50 {if(!IsCap(prod[j][k]))
51 {e=0;
52 add(first[n],prod[j][k]);
53 }else
54 {n1 = findpos(prod[j][k]);
55 addarr(first[n],first[n1]);
56 if(eps[n1] == 0)
57 e=0;
58 } }if(e==1)
59 eps[n]=1;}}}}
60 void findfollow()
61 {int i,j,k,n,e,n1;
62 n = findpos(prod[0][0]);
63 add(follow[n], '#');
64 for(i=0;i<count;i++)
65 {for(j=0;j<count;j++)
66 {k = strlen(prod[j])-1;
67 for(k>0;k--
68 {if(IsCap(prod[j][k]))
69 {n=findpos(prod[j][k]);
70 {n=findpos(prod[j][k]);
71 if(prod[j][k+1] == '\0') // A -> aB
72 {n1 = findpos(prod[j][0]);
73 addarr(follow[n],follow[n1]);
74 }if(IsCap(prod[j][k+1])) // A -> aBb
75 {n1 = findpos(prod[j][k+1]);
76 addarr(follow[n],first[n1]);
77 if(eps[n1]==1)
78 {n1=findpos(prod[j][0]);
79 addarr(follow[n],follow[n1]);}}
80 else if(prod[j][k+1] != '\0')
81 add(follow[n],prod[j][k+1]);}}}}
82 int main()
83 {char s[max],i;
84 printf("\nEnter the productions(type '\0' to end)\n");
85 scanf("%s",s);
86 while(strcmp("end",s)){
87 addprod(s);
88 scanf("%s",s);
89 findfirst();
90 findfollow();
91 for(i=0;i<strlen(nt);i++)
92 {printf("%c\t",nt[i]);
93 printf("%s",first[i]);
94 if(eps[i]==1)
95 printf("%c\t", (char)238);
96 else
97 printf("\t");
98 printf("%s\n", follow[i]);
99 getch();
100 }
```

OUTPUT:



The screenshot shows the Visual Studio Code interface. The terminal window displays the output of a C++ program. The program prompts the user to enter productions, followed by a series of inputs: E->E+T, E->T, T->T*F, T->F, F->(E), F->i, and end. The program then outputs a list of productions: E (i #+), T (i #+*), and F (i #+*). The Explorer window on the right shows the file structure of the project, including a folder named LABS and a file named week_5_first.cpp.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS> cd "c:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS\CD LABS\" ; if ($?) { g++ week_5_first.cpp -o wee
k_5_first } ; if ($?) { .\week_5_first }

Enter the productions(type 'end' at the last of the production)
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
end
E      (i      #+)
T      (i      #+*)
F      (i      #+*)
[]
```

RESULT :

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

PREDICTIVE PARSING

EXPERIMENT-6

Aim: A program for Predictive Parsing

Algorithm:-

1. Start the program.
2. Initialize the required variables.
3. Get the number of coordinates and productions from the user.
4. Perform the following
for (each production $A \rightarrow \alpha$ in G) {
for (each terminal a in $FIRST(\alpha)$)
add $A \rightarrow \alpha$ to $M[A, a]$;
if (ϵ is in $FIRST(\alpha)$)
for (each symbol b in $FOLLOW(A)$)
add $A \rightarrow \alpha$ to $M[A, b]$;
5. Print the resulting stack.
6. Print if the grammar is accepted or not.
7. Exit the program.

CODE :

```
File Edit Selection View Go Run Terminal Help
week_5_predic.cpp - LABS - Visual Studio Code

week_5_first.cpp  week_5_predic.cpp  week_5_predic.cpp  week_5_predic.cpp

1 //nikhil kumar singh -ra1911033010102
2 #include<stdio.h>
3 #include<conio.h>
4 #include<string.h>
5 int main()
6 {
7     char fin[10][20], st[10][20], ft[20][20], fol[20][20];
8     int a=0, e, i, t, b, c, n, k, l=0, j, s, m, p;
9     printf("enter the no. of productions\n");
10    scanf("%d", &n);
11    printf("enter the productions in a grammar\n");
12    for(i=0; i<n; i++)
13        scanf("%s", st[i]);
14    for(i=0; i<n; i++)
15        fol[i][0] = '\0';
16    for(s=0; s<n; s++)
17    {
18        for(i=0; i<n; i++)
19        {
20            j=3;
21            l=0;
22            a=0;
23            l1: if(!((st[i][j]>64)&&(st[i][j]<91)))
24            {
25                for(m=0; m<l; m++)
26                {
27                    if(ft[i][m]==st[i][j])
28                        goto s1;
29                }
30                ft[i][l]=st[i][j];
31                l=l+1;
32                s1: j=j+1;
33            }
34        }
35    }
36    a++;
37    }
38    b=0;
39    while(ft[a][b]!='\0')
40    {
41        for(m=0; m<l; m++)
42        {
43            if(ft[i][m]==ft[a][b])
44                goto s2;
45        }
46        ft[i][l]=ft[a][b];
47        l=l+1;
48        s2: b=b+1;
49    }
50    while(st[i][j]!='\0')
51    {
52        if(st[i][j]=='|')
53        {
54            j=j+1;
55            goto l1;
56        }
57        if(st[i][j]!='|')
58        {
59            j=j+1;
60            goto l1;
61        }
62        j=j+1;
63        ft[i][l]=st[i][j];
64        l=l+1;
65    }
66    printf("first pos\n");
67    for(i=0; i<n; i++)
68        printf("FIRS[%c]=%s\n", st[i][0], ft[i]);
69    fol[0][0]='$';
70    for(i=0; i<n; i++)
71    {
72        i=0;
73        k1: while((st[i][0]!=st[k][j])&&(k<n))
74        {
75            if(st[k][j]!='\0')
76            {
77                k++;
78                j=2;
79            }
80            j=j+1;
81            if(st[i][0]==st[k][j-1])
82            {
83                if((st[k][j]!='|')&&(st[k][j]!='\0'))
84                {
85                    a=0;
86                    if(!((st[k][j]>64)&&(st[k][j]<91)))
87                    {
88                        for(m=0; m<l; m++)
89                        {
90                            if(fol[i][m]==st[k][j])
91                                goto q3;
92                        }
93                        fol[i][l]=st[k][j];
94                        l++;
95                        q3: p++;
96                    }
97                    else
98                    {
99                        while(st[k][j]!=st[a][0])
100                        {
101                            a++;
102                        }
103                        p=0;
104                    }
105                }
106            }
107        }
108    }
109    }
```

OUTPUT:

The screenshot shows the Visual Studio Code interface. The terminal window at the bottom displays the following commands and output:

```

PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS> cd "c:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS\CD LABS\" ; if (
$?) { g++ week_5_predic.cpp -o week_5_predic } ; if ($?) { .\week_5_predic }
enter the no. of productions
2
enter the productions in a grammar
S->CC
C->cC|d
first pos
FIRS[S]=cd
FIRS[C]=cd
follow pos
FOLLOW[S]=$
FOLLOW[C]=cd$

M[S,c]=S->CC
M[S,d]=S->CC
M[C,c]=C->cC
M[C,d]=C->d
PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS\CD LABS>

```

The Explorer view on the right shows the project structure:

- LABS
 - .vscode
 - AI LABS
 - CD LABS
 - PROG.txt
 - programm.txt
 - tempCodeRunnerFil...
 - week_1.cpp
 - week_1.exe
 - week_2.c
 - week_2.exe
 - week_3.py
 - week_4_1f.cpp
 - week_4_1f.exe
 - week_4_1r.cpp
 - week_4_1r.exe
 - week_5_first.cpp
 - week_5_first.exe
 - week_5_predic.cpp
 - week_5_predic.exe
 - week_6_shift.exe
 - week_6_shift.py
 - week_lead.cpp
 - week_lead.exe
 - week_lr0.cpp
 - week_lr0.exe
 - week_nostfix.nu...

Result:-

The program was successfully compiled and run

Shift Reduce

Aim: A program to implement Shift Reduce

Shift Reduce parser attempts for the construction of parse in a similar manner as done in bottom up parsing i.e. the parse tree is constructed from leaves(bottom) to the root(up). A more general form of shift reduce parser is LR parser.

This parser requires some data structures i.e.

- A input buffer for storing the input string.
- A stack for storing and accessing the production rules.

Basic Operations –

- Shift: This involves moving of symbols from input buffer onto the stack.
- Reduce: If the handle appears on top of the stack then, its reduction by using appropriate production rule is done i.e. RHS of production rule is popped out of stack and LHS of production rule is pushed onto the stack.
- Accept: If only start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accept action is obtained, it means successful parsing is done.
- Error: This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action.

CODE :

```
File Edit Selection View Go Run Terminal Help
week_6_shift.py - LABS - Visual Studio Code

week_6_shift.py ×
1 #Shift reduce parsing
2 from collections import defaultdict
3
4 n = int(input('Enter the number of production rules: '))
5 (variable) memo: defaultdict[Any, list]
6 memo = defaultdict(list)
7
8 startsymbol = ''
9 print('Enter the production rules:')
10 for i in range(n):
11     inp = input()
12     left, _, right = inp.partition('->')
13     left = left.strip()
14     right = right.strip()
15     lexemes = right.split('|')
16     if i == 0:
17         start_symbol = left
18     for c in lexemes:
19         memo[left].append(c)
20
21
22 inp = input('Enter the input string: ')
23 inp = inp + '$'
24 i = 0
25 s = '$'
26 print(f'{"Stack Contents": <15>' + " | " + f'{"Input Buffer": <15>' + " | " + f'Par')
27 accepted = True
28 action = ''
29 while i < len(inp) - 1 or len(s) != 2:
30     action = ''
31     found = False
32     for key, values in memo.items():
33         for value in values:
34             if len(value) <= len(s) and value == s[(len(s) - len(value))]:
35                 action = 'Reduce ' + key + ' -> ' + value
36                 print(f's: <15>' + " | " + f'{inp[i:]: <15>' + " | " + f'{action}')
37                 s = s[:-len(value)]
38                 s = s + key
39                 found = True
40
41 if not found and i < len(inp):
42     action = 'Shift'
43     print(f's: <15>' + " | " + f'{inp[i:]: <15>' + " | " + f'{action}')
44     s = s + inp[i]
45     i += 1
46
47 if action == '':
48     accepted = False
49     break
50
51 if accepted:
52     print(f's: <15>' + " | " + f'{inp[i:1]: <15>' + " | " + f'Accepted')
53 else:
54     print(f's: <15>' + " | " + f'{inp[i:]: <15>' + " | " + f'Rejected')
55
56
```


OUTPUT:

The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the execution of a Python script named `week_6_shift.py`. The script prompts the user to enter the number of production rules (3) and the production rules (`E -> E+E`, `E -> E*i`, `E -> i`). It then prompts for an input string (`i+i*i`) and displays a table of LR(0) item sets and their transitions.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS> & "C:/Users/Nikhil Kumar Singh/AppData/Local/Programs/Python/Python39/python.exe" "c:/Users/Nikhil Kumar Singh/Desktop/New folder/LABS/CD LABS/week_6_shift.py"
Enter the number of production rules: 3
Enter the production rules:
E->E+E
E->E*i
E->i
Enter the input string: i+i*i
Stack Contents | Input Buffer | Parsing Action
$              | i+i*i$      | Shift
$i             | +i*i$       | Reduce E -> i
$E            | +i*i$       | Shift
$E+           | i*i$        | Shift
$E+i          | *i$         | Reduce E -> i
$E+E          | *i$         | Reduce E -> E+E
$E            | *i$         | Shift
$E*           | i$          | Shift
$E*i          | $           | Reduce E -> i
$E*iE         | $           | Reduce E -> E*iE
$E            |             | Accepted
PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS> 
```

The Explorer pane on the right shows the file structure of the project, including files like `PROG.txt`, `programm.txt`, `tempCodeRunnerFil...`, `week_1.cpp`, `week_1.exe`, `week_2.c`, `week_2.exe`, `week_3.py`, `week_4_LF.cpp`, `week_4_LF.exe`, `week_4_Lr.cpp`, `week_4_Lr.exe`, `week_5_first.cpp`, `week_5_first.exe`, `week_5_predic.cpp`, `week_5_predic.exe`, `week_6_shift.exe`, `week_6_shift.py`, `week_lead.cpp`, `week_lead.exe`, `week_lr0.cpp`, `week_lr0.exe`, `week_postfix.py`, and `week_quad.c`.

RESULT :

The program was successfully compiled and run.

LEADING AND TRAILING

LAB EXP 8

AIM : A program to implement Leading and Trailing

ALGORITHM :

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

The screenshot displays the Visual Studio Code interface with a C++ file named `week_lead.cpp` open. The code implements the Leading and Trailing algorithms for a grammar. The `main` function prompts the user to enter the number of variables, terminals, and productions, then reads the grammar rules. The `leading()` function recursively checks for the first non-terminal in the left-hand side of a production and prints the result. The `trailing()` function (partially visible) would perform a similar check from the right-hand side. The Explorer panel on the right shows the project structure, including various lab files and executables.

```
1 // nikhil kumar singh - RA1911033010102
2 #include<iostream>
3 #include<conio.h>
4 #include<stdio.h>
5 #include<string.h>
6 #include<stdlib.h>
7 using namespace std;
8
9 int vars, terms, i, j, k, m, rep, count, temp = -1;
10 char var[10], term[10], lead[10][10], trail[10][10];
11 struct grammar
12 {
13     int prodno;
14     char lhs, rhs[20][20];
15 } gram[50];
16
17 void get()
18 {
19     cout<<"\nLEADING AND TRAILING\n";
20     cout<<"\nEnter the no. of variables : ";
21     cin>>vars;
22     cout<<"\nEnter the variables : \n";
23     for(i=0; i<vars; i++)
24     {
25         cin>>gram[i].lhs;
26         var[i] = gram[i].lhs;
27     }
28     cout<<"\nEnter the no. of terminals : ";
29     cin>>terms;
30     cout<<"\nEnter the terminals : ";
31     for(j=0; j<terms; j++)
32         cin>>term[j];
33     cout<<"\nPRODUCTION DETAILS\n";
34
35     for(i=0; i<vars; i++)
36     {
37         cout<<gram[i].lhs<<"->";
38         cin>>gram[i].rhs[j];
39     }
40 }
41
42 void leading()
43 {
44     for(i=0; i<vars; i++)
45     {
46         for(j=0; j<gram[i].prodno; j++)
47         {
48             for(k=0; k<terms; k++)
49             {
50                 if(gram[i].rhs[j][0] == term[k])
51                     lead[i][k] = 1;
52                 else
53                 {
54                     if(gram[i].rhs[j][1] == term[k])
55                         lead[i][k] = 1;
56                 }
57             }
58         }
59     }
60
61     for(rep=0; rep<vars; rep++)
62     {
63         for(i=0; i<vars; i++)
64         {
65             for(j=0; j<gram[i].prodno; j++)
66             {
67                 for(m=1; m<vars; m++)
68                 {
69                     if(gram[i].rhs[j][0] == var[m])
70                         // ...
71                 }
72             }
73         }
74     }
75 }
```

File Edit Selection View Go Run Terminal Help

week_lead.cpp - LABS - Visual Studio Code

week_lead.cpp

72

if(gram[i].rhs[j][0]==var[m])

73

{

74

temp=m;

75

goto out;

76

}

77

out:

78

for(k=0;k<terms;k++)

79

{

80

if(lead[temp][k]==1)

81

lead[i][k]=1;

82

}

83

}

84

void trailing()

85

{

86

for(i=0;i<vars;i++)

87

{

88

for(j=0;j<gram[i].prodno;j++)

89

{

90

count=0;

91

while(gram[i].rhs[j][count]!='\x0')

92

count++;

93

for(k=0;k<terms;k++)

94

{

95

if(gram[i].rhs[j][count-1]==term[k])

96

trail[i][k]=1;

97

else

98

{

99

if(gram[i].rhs[j][count-2]==term[k])

100

trail[i][k]=1;

101

}

102

}

103

for(rep=0;rep<vars;rep++)

104

{

105

for(i=0;i<vars;i++)

106

{

107

for(j=0;j<gram[i].prodno;j++)

108

{

109

count=0;

110

while(gram[i].rhs[j][count]!='\x0')

111

count++;

112

for(m=1;m<vars;m++)

113

{

114

if(gram[i].rhs[j][count-1]==var[m])

115

temp=m;

116

}

117

for(k=0;k<terms;k++)

118

{

119

if(trail[temp][k]==1)

120

trail[i][k]=1;

121

}

122

}

123

}

124

}

125

}

126

}

127

}

128

}

129

}

130

}

131

}

132

}

133

}

134

}

135

int main()

136

{

137

get();

138

leading();

139

trailing();

140

display();

141

}

LABS

CD LABS

PROG.txt

programm.txt

tempCodeRunnerFil...

week_1.cpp

week_1.exe

week_2.c

week_2.exe

week_3.py

week_4_LF.cpp

week_4_LF.exe

week_4_Lr.cpp

week_4_Lr.exe

week_5_first.cpp

week_5_first.exe

week_5_predic.cpp

week_5_predic.exe

week_6_shift.exe

week_6_shift.py

week_lead.cpp

week_lead.exe

week_lr0.cpp

week_lr0.exe

week_postfix.py

week_quad.c

week_quad.exe

OUTPUT:

File Edit Selection View Go Run Terminal Help

week_lead.cpp - LABS - Visual Studio Code

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL

Code + -

EXPLORER

PS C:\Users\Wikhil Kumar Singh\Desktop\New folder\LABS\CD LABS> cd "c:\Users\Wikhil Kumar Singh\Desktop\New folder\LABS\CD LABS\" ; if (\$?) { g++ week_lead.cpp -o week_lead } ; if (\$?) { .\week_lead }

LEADING AND TRAILING

Enter the no. of variables : 3

Enter the variables :
E

PRODUCTION DETAILS

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

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

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

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

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

PS C:\Users\Wikhil Kumar Singh\Desktop\New folder\LABS\CD LABS>

LABS

CD LABS

PROG.txt

programm.txt

tempCodeRunnerFil...

week_1.cpp

week_1.exe

week_2.c

week_2.exe

week_3.py

week_4_LF.cpp

week_4_LF.exe

week_4_Lr.cpp

week_4_Lr.exe

week_5_first.cpp

week_5_first.exe

week_5_predic.cpp

week_5_predic.exe

week_6_shift.exe

week_6_shift.py

week_lead.cpp

week_lead.exe

week_lr0.cpp

week_lr0.exe

week_postfix.py

week_quad.c

week_quad.exe

RESULT :

The program was successfully compiled and run.

Computation of LR(0) Items

Aim: A program to implement LR(0) items

Algorithm:-1. Start.

2. Create structure for production with LHS and RHS.

3. Open file and read input from file.

4. Build state 0 from extra grammar Law $S' \rightarrow S \$$ 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.

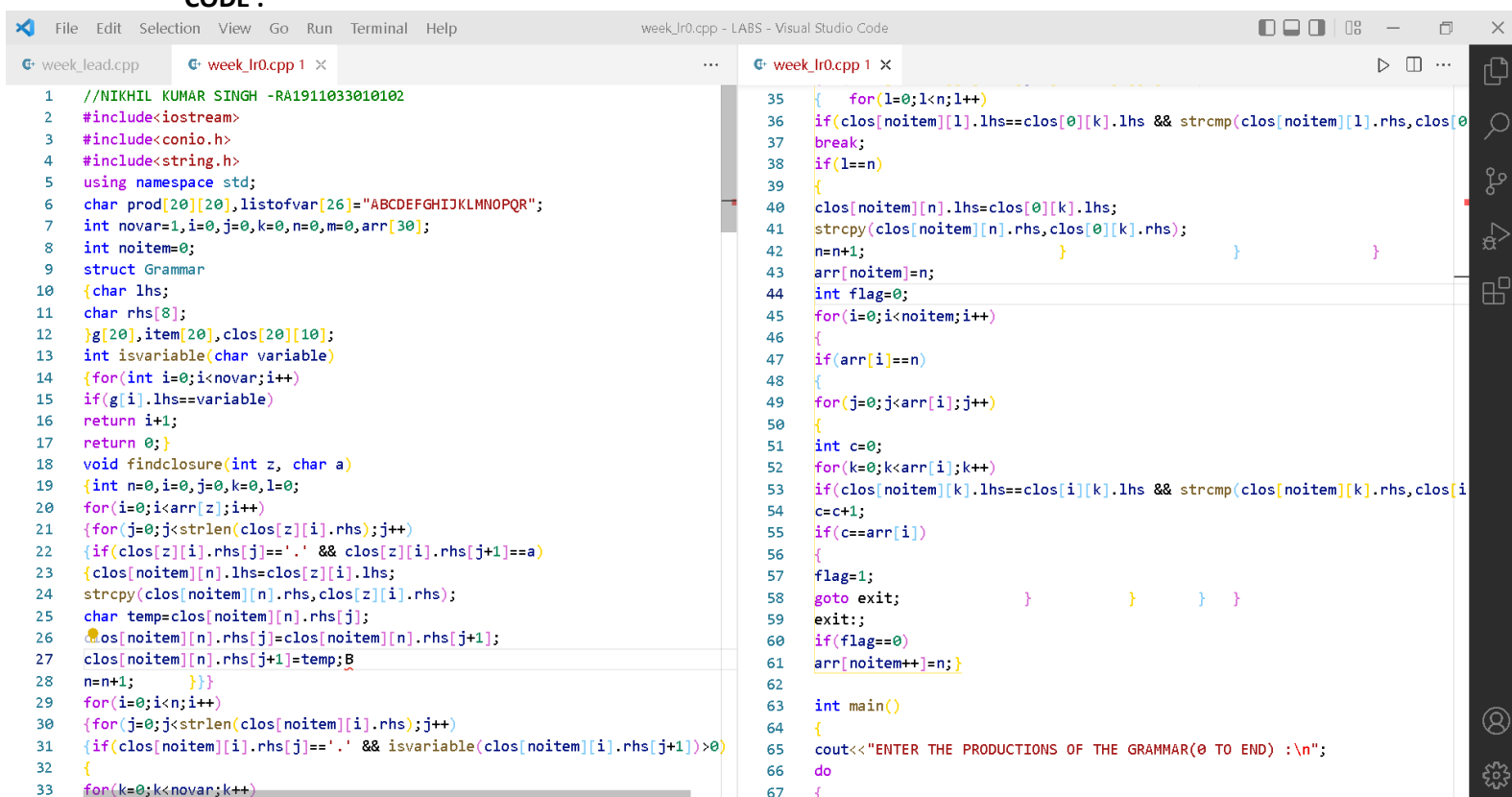
10. Go to step 5.

11. End of state building.

12. Display the output.

13. End.

CODE :



```
1 //NIKHIL KUMAR SINGH -RA1911033010102
2 #include<iostream>
3 #include<conio.h>
4 #include<string.h>
5 using namespace std;
6 char prod[20][20],listofvar[26]="ABCDEFGHJKLMNOPQR";
7 int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
8 int noitem=0;
9 struct Grammar
10 {char lhs;
11 char rhs[8];
12 }g[20],item[20],clos[20][10];
13 int isvariable(char variable)
14 {for(int i=0;i<novar;i++)
15 if(g[i].lhs==variable)
16 return i+1;
17 return 0;}
18 void findclosure(int z, char a)
19 {int n=0,i=0,j=0,k=0,l=0;
20 for(i=0;i<arr[z];i++)
21 {for(j=0;j<strlen(clos[z][i].rhs);j++)
22 {if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==a)
23 {clos[noitem][n].lhs=clos[z][i].lhs;
24 strcpy(clos[noitem][n].rhs,clos[z][i].rhs);
25 char temp=clos[noitem][n].rhs[j];
26 clos[noitem][n].rhs[j]=clos[noitem][n].rhs[j+1];
27 clos[noitem][n].rhs[j+1]=temp;B
28 n=n+1; }}}
29 for(i=0;i<n;i++)
30 {for(j=0;j<strlen(clos[noitem][i].rhs);j++)
31 {if(clos[noitem][i].rhs[j]=='.' && isvariable(clos[noitem][i].rhs[j+1])>0)
32 {
33 for(k=0;k<novar;k++)
34 {
35 for(l=0;l<n;l++)
36 if(clos[noitem][l].lhs==clos[0][k].lhs && strcmp(clos[noitem][l].rhs,clos[0][k].rhs)==0)
37 break;
38 if(l==n)
39 {
40 clos[noitem][n].lhs=clos[0][k].lhs;
41 strcpy(clos[noitem][n].rhs,clos[0][k].rhs);
42 n=n+1;
43 }
44 }
45 arr[noitem]=n;
46 int flag=0;
47 for(i=0;i<noitem;i++)
48 {
49 if(arr[i]==n)
50 {
51 int c=0;
52 for(k=0;k<arr[i];k++)
53 if(clos[noitem][k].lhs==clos[i][k].lhs && strcmp(clos[noitem][k].rhs,clos[i][k].rhs)==0)
54 c=c+1;
55 if(c==arr[i])
56 {
57 flag=1;
58 goto exit;
59 }
60 }
61 if(flag==0)
62 arr[noitem++]=n;
63 }
64 int main()
65 {
66 cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n";
67 do
68 {
```

The image shows a screenshot of the Visual Studio Code editor with two C++ files open. The left file, `week_lead.cpp`, contains a `main` function that reads grammar rules from standard input and computes LR(0) items and transitions. The right file, `week_lr0.cpp`, contains a function to compute the LR(0) closure for a given item.

```
68 cin>>prod[i++];
69 }while(strcmp(prod[i-1],"0")!=0);
70 for(n=0;n<i-1;n++)
71 {
72     m=0;
73     j=nowar;
74     g[nowar++].lhs=prod[n][0];
75     for(k=3;k<strlen(prod[n]);k++)
76     {
77         if(prod[n][k] != '|')
78             g[j].rhs[m++]=prod[n][k];
79         if(prod[n][k]=='|')
80         {
81             g[j].rhs[m]='\0';
82             m=0;
83             j=nowar;
84             g[nowar++].lhs=prod[n][0];
85             for(i=0;i<26;i++)
86                 if(!isvariable(listofvar[i]))
87                     break;
88             g[0].lhs=listofvar[i];
89             char temp[2]={g[1].lhs,'\0'};
90             strcat(g[0].rhs,temp);
91             cout<<"\n\n augmented grammar \n";
92             for(i=0;i<nowar;i++)
93                 cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";
94
95             for(i=0;i<nowar;i++)
96             {
97                 clos[noitem][i].lhs=g[i].lhs;
98                 strcpy(clos[noitem][i].rhs,g[i].rhs);
99                 if(strcmp(clos[noitem][i].rhs,"e")==0)
100                     strcpy(clos[noitem][i].rhs,".");
```

```
100 strcpy(clos[noitem][i].rhs,".");
101 else
102 {
103     for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)
104         clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1];
105         clos[noitem][i].rhs[0]='.';
106         arr[noitem++]=nowar;
107         for(int z=0;z<noitem;z++)
108         {
109             char list[10];
110             int l=0;
111             for(j=0;j<arr[z];j++)
112             {
113                 for(k=0;k<strlen(clos[z][j].rhs)-1;k++)
114                 {
115                     if(clos[z][j].rhs[k]=='.')
116                     {
117                         for(m=0;m<l;m++)
118                             if(list[m]==clos[z][j].rhs[k+1])
119                                 break;
120                         if(m==l)
121                             list[l++]=clos[z][j].rhs[k+1];
122                         for(int x=0;x<l;x++)
123                             findclosure(z,list[x]);
124                         cout<<"\n THE SET OF ITEMS ARE \n\n";
125                         for(int z=0; z<noitem; z++)
126                         {
127                             cout<<"\n I"<<z<<"\n\n";
128                             for(j=0;j<arr[z];j++)
129                                 cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";
```

OUTPUT:

The image shows a Visual Studio Code editor window with a C++ program in the main editor and a file explorer on the right.

Visual Studio Code Interface:

- Top Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Left Panel:** DEBUG CONSOLE, PROBLEMS, OUTPUT, TERMINAL.
- Right Panel:** EXPLORER, OUTLINE, TIMELINE.

Code Editor Content:

```
PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS\CD LABS> cd "c:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS\CD LABS\" ; if ($?) { g++ week_lr0.cpp -o week_lr0 } ; if ($?) { .\week_lr0 }
```

ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :

E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
0

augmented grammar

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

I0

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

I1

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

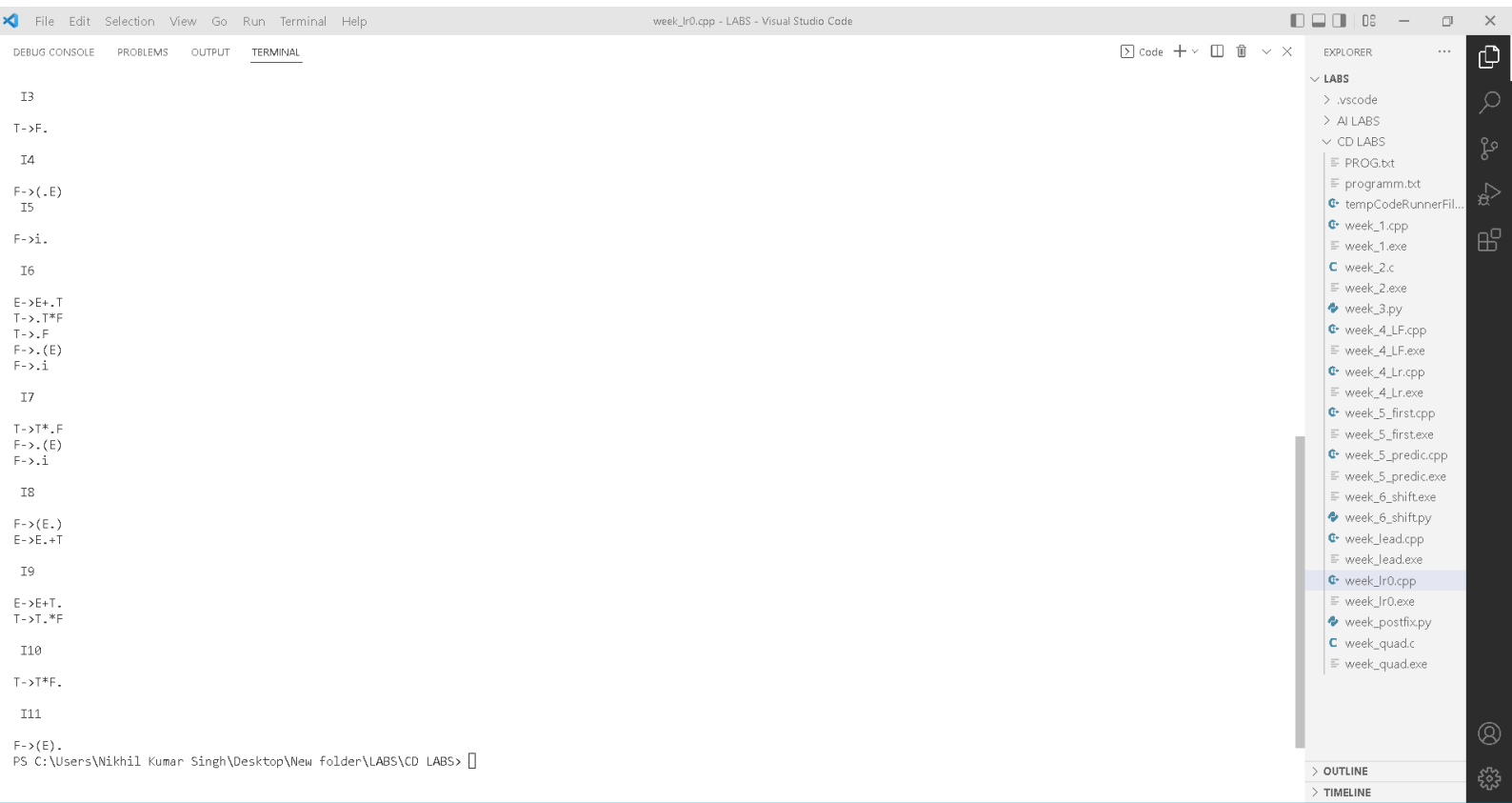
I2

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

I3

File Explorer Content:

- LABS
 - .vscode
 - AI LABS
 - CD LABS
 - PROG.txt
 - programm.txt
 - tempCodeRunnerFill...
 - week_1.cpp
 - week_1.exe
 - week_2.c
 - week_2.exe
 - week_3.py
 - week_4_LF.cpp
 - week_4_LF.exe
 - week_4_Lr.cpp
 - week_4_Lr.exe
 - week_5_first.cpp
 - week_5_first.exe
 - week_5_predic.cpp
 - week_5_predic.exe
 - week_6_shift.exe
 - week_6_shift.py
 - week_lead.cpp
 - week_lead.exe
 - week_lr0.cpp
 - week_lr0.exe
 - week_postfix.py
 - week_quad.c
 - week_quad.exe



RESULT :

The program was successfully compiled and run.

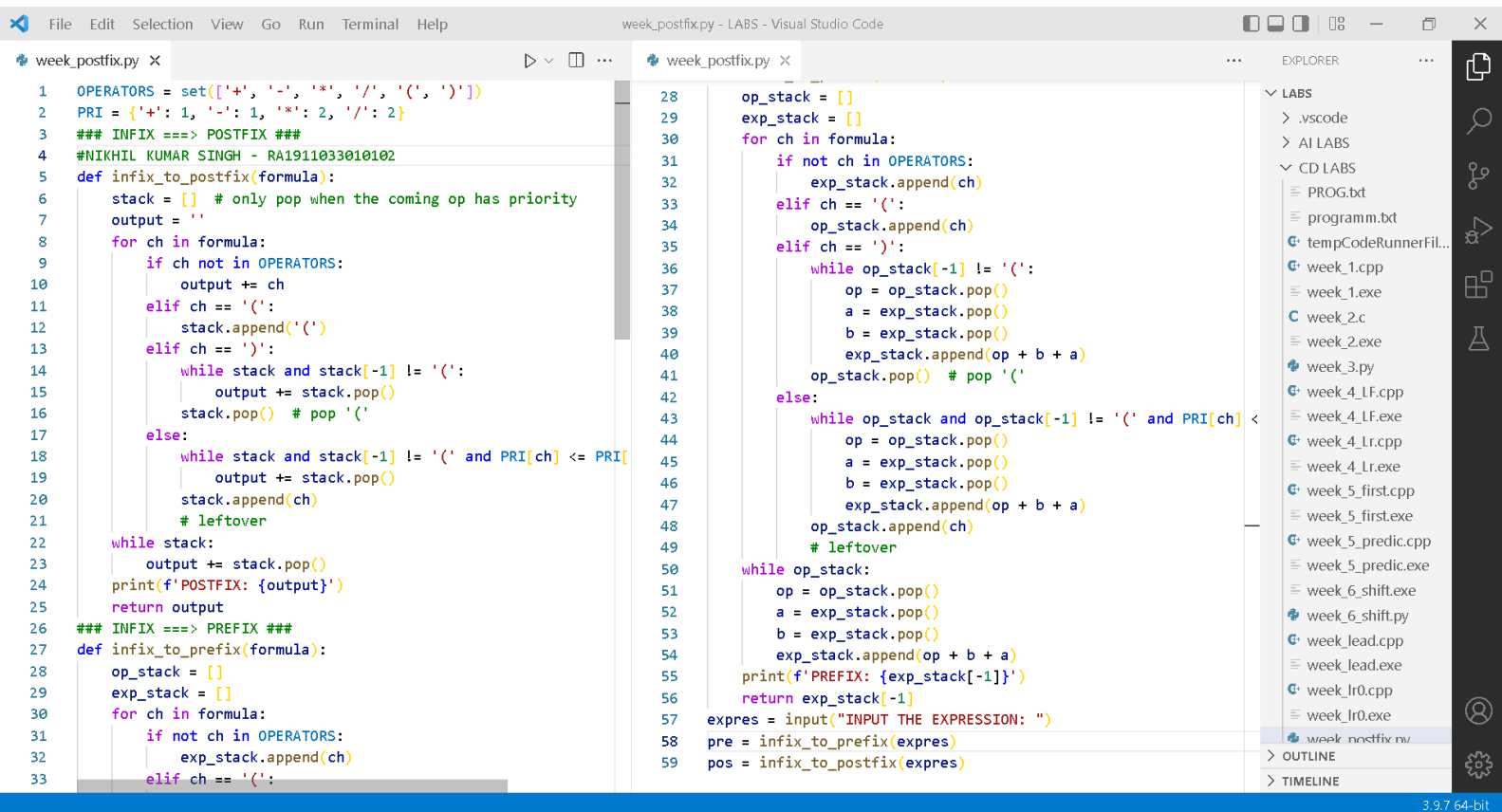
Postfix, Prefix

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

Algorithm:-

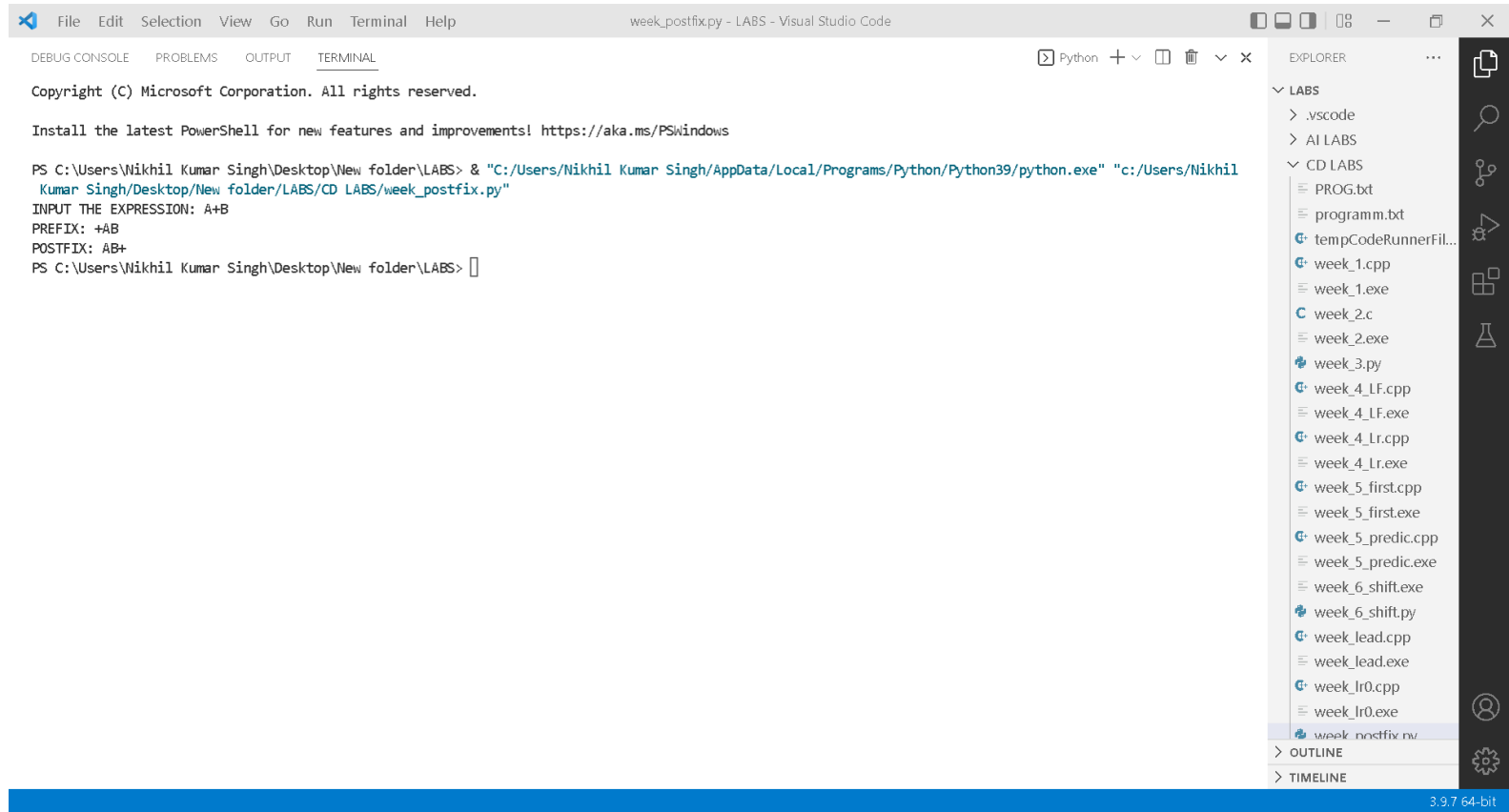
1. Declare set of operators.
2. Initialize an empty stack.
3. To convert INFIX to POSTFIX follow the following steps
4. Scan the infix expression from left to right.
5. If the scanned character is an operand, output it.
6. Else, If the precedence of the scanned operator is greater than the precedence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

CODE :



```
1 OPERATORS = set(['+', '-', '*', '/', '(', ')'])
2 PRI = {'+': 1, '-': 1, '*': 2, '/': 2}
3 ### INFIX ==> POSTFIX ###
4 #NIKHIL KUMAR SINGH - RA1911033010102
5 def infix_to_postfix(formula):
6     stack = [] # only pop when the coming op has priority
7     output = ''
8     for ch in formula:
9         if ch not in OPERATORS:
10             output += ch
11         elif ch == '(':
12             stack.append('(')
13         elif ch == ')':
14             while stack and stack[-1] != '(':
15                 output += stack.pop()
16             stack.pop() # pop '('
17         else:
18             while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
19                 output += stack.pop()
20             stack.append(ch)
21             # leftover
22     while stack:
23         output += stack.pop()
24     print(f'POSTFIX: {output}')
25     return output
26 ### INFIX ==> PREFIX ###
27 def infix_to_prefix(formula):
28     op_stack = []
29     exp_stack = []
30     for ch in formula:
31         if not ch in OPERATORS:
32             exp_stack.append(ch)
33         elif ch == '(':
34             op_stack.append(ch)
35         elif ch == ')':
36             while op_stack[-1] != '(':
37                 op = op_stack.pop()
38                 a = exp_stack.pop()
39                 b = exp_stack.pop()
40                 exp_stack.append(op + b + a)
41             op_stack.pop() # pop '('
42         else:
43             while op_stack and op_stack[-1] != '(' and PRI[ch] < PRI[op_stack[-1]]:
44                 op = op_stack.pop()
45                 a = exp_stack.pop()
46                 b = exp_stack.pop()
47                 exp_stack.append(op + b + a)
48             op_stack.append(ch)
49             # leftover
50     while op_stack:
51         op = op_stack.pop()
52         a = exp_stack.pop()
53         b = exp_stack.pop()
54         exp_stack.append(op + b + a)
55     print(f'PREFIX: {exp_stack[-1]}')
56     return exp_stack[-1]
57 expres = input("INPUT THE EXPRESSION: ")
58 pre = infix_to_prefix(expres)
59 pos = infix_to_postfix(expres)
```

OUTPUT:



```
PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS> & "C:/Users/Nikhil Kumar Singh/AppData/Local/Programs/Python/Python39/python.exe" "c:/Users/Nikhil Kumar Singh/Desktop/New folder/LABS/CD LABS/week_postfix.py"
INPUT THE EXPRESSION: A+B
PREFIX: +AB
POSTFIX: AB+
PS C:\Users\Nikhil Kumar Singh\Desktop\New folder\LABS>
```


RESULT :

The program was successfully compiled and run.