

Ex-1. LEXICAL ANALYZER

**S. AKANSHA
RA1911029010060
CSE-CN P1**

AIM:

To write a program to implement a lexical analyzer in C.

ALGORITHM:

- Start.
- Get the input program from the file sentence.txt.
- Read the program line by line and check if each word in a line is a keyword, identifier, math operator, logical operator, numerical value or others symbol.
- For each lexeme read, generate a token as follows:
 - If the lexeme is a keyword, then the token is the keyword itself.
 - If the lexeme is an identifier, then the token generated is printed on the console as identifier.
 - In the same way the math operator, logical operator, numerical values and others symbol are printed on the console.
- The stream of tokens generated are displayed in the console output.
- Stop.

PROGRAM:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
void keyw(char *p);
int i=0,id=0,kw=0,num=0,op=0;
char keys[32][10]={ "auto","break","case","char","const","continue","default",
"do","double","else","enum","extern","float","for","goto",
"if","int","long","register","return","short","signed",
"sizeof","static","struct","switch","typedef","union",
"unsigned","void","volatile","while"};
int main()
{
char ch,str[25],seps[15] = " \t\n;(){}[]#<>",oper[]="!%^&*-=~|.<>/?";
int j;
char fname[50];
FILE *f1;
printf("enter file path (drive:\\fold\\filename)\n");
scanf("%s",fname);
f1 = fopen(fname,"r");
if(f1==NULL)
{
printf("file not found");
exit(0);
}
while((ch=fgetc(f1))!=EOF)
{
for(j=0;j<=14;j++)
```

```

{
if(ch==oper[j])
{
printf("%c is an operator\n",ch);
op++;
str[i]='\0';
keyw(str);
}
for(j=0;j<=14;j++)
{
if(i==-1)
break;
if(ch==seps[j])
{
if(ch=='#')
{
while(ch!='>')
{
printf("%c",ch);
ch=fgetc(f1);
}
printf("%c is a header file\n",ch);
i=-1;
break;
}
if(ch=="")
{
do
{
ch=fgetc(f1);
printf("%c",ch);
}while(ch!="");
printf("\b is an argument\n");
i=-1;
break;
}
str[i]='\0';
keyw(str);
}
if(i!= -1)
{
str[i]=ch;
i++;
}
else
i=0;
}
printf("Keywords: %d \n",kw);
printf("Identifiers: %d\n",id);
printf("Operators: %d\n",op);
printf("Numbers: %d\n",num);
}
void keyw(char *p)
{
int k,flag=0;

```

```

for(k=0;k<=31;k++)
{
if(strcmp(keys[k],p)==0)
{
printf("%s is a keyword\n",p);
kw++;
flag=1;
break;
}
if(flag==0)
{
if(isdigit(p[0]))
{
printf("%s is a number\n",p);
num++;
}
else
{
if(p[0]!='\0')
{
printf("%s is an identifier\n",p);
id++;
}}}
i=-1;
}
/*
8
9 #include<stdio.h>
10 #include<ctype.h>
11 #include<string.h>
12 #include<stdlib.h>
13 void keyw(char *p);
14 int i=0,id=0,kw=0,num=0,op=0;
15 char keys[32][10]={"auto","break","case","char","const","continue","default",
16 "do","double","else","enum","extern","float","for","goto",
17 "if","int","long","register","return","short","signed",
18 "sizeof","static","struct","switch","typedef","union",
19 "unsigned","void","volatile","while"};
20 int main()
21 {
22 char ch,str[25],seps[15]="\t\n;(){}[]#<>,&operator[]=%^&*-+=~|.<>/?";
23 int j;
24 char fname[50];
25 FILE *f1;
26 printf("enter file path (drive:\\fold\\filename)\n");
27 scanf("%s",fname);
28 f1 = fopen(fname,"r");
29 if(f1==NULL)
30 {
31 printf("file not found");
32 exit(0);
33 }
34 while((ch=fgetc(f1))!=EOF)
35 {
36 for(j=0;j<=14;j++)
37 {
38 if(ch==oper[j])
39 {
40 printf("%c is an operator\n",ch);
41 op++;
42 str[i]='\0';

```



```
42 str[i]='\0';
43 keyw(str);
44 }
45 for(j=0;j<=14;j++)
46 {
47 if(i==-1)
48 break;
49 if(ch==seps[j])
50 {
51 if(ch=='#')
52 {
53 while(ch!='>')
54 {
55 printf("%c",ch);
56 ch=fgetc(f1);
57 }
58 printf("%c is a header file\n",ch);
59 i=-1;
60 break;
61 }
62 if(ch=='"')
63 {
64 do
65 {
66 ch=fgetc(f1);
67 printf("%c",ch);
68 }while(ch!="");
69 printf("\b is an argument\n");
70 i=-1;
71 break;
72 }
73 str[i]='\0';
74 keyw(str);
75 }
76 if(i!=-1)
77 f
```

input

Press ENTER to exit console []

9:30 PM
1/17/2022

```
81 else
82 i=0;
83 }
84 printf("Keywords: %d \n",kw);
85 printf("Identifiers: %d\n",id);
86 printf("Operators: %d\n",op);
87 printf("Numbers: %d\n",num);
88 }
89 void keyw(char *p)
90 [
91 int k,flag=0;
92 for(k=0;k<=31;k++)
93 {
94 if(strcmp(keys[k],p)==0)
95 {
96 printf("%s is a keyword\n",p);
97 kw++;
98 flag=1;
99 break;
100 }
101 if(flag==0)
102 {
103 if(isdigit(p[0]))
104 {
105 printf("%s is a number\n",p);
106 num++;
107 }
108 else
109 {
110 if(p[0]!='\0')
111 {
112 printf("%s is an identifier\n",p);
113 id++;
114 }
115 i=-1;
116 }
```

input

Press ENTER to exit console []

9:30 PM
1/17/2022

Sentence.txt

```
#include <stdio.h>
int main()
{
printf("Hello have a great day");
return 0;
}
```

The screenshot shows the OnlineGDB beta IDE interface. On the left, there's a sidebar with options like 'code. compile. run. debug. share.', 'IDE', 'My Projects', and 'Classroom'. The main area has two tabs: 'main.c' and 'sentence.txt'. The 'main.c' tab contains the following C code:

```
1 #include <stdio.h>
2 int main()
3 {
4     printf("Hello have a great day");
5     return 0;
6 }
```

OUTPUT:

enter file path (drive:\fold\filename)

sentence.txt

#include <stdio.h> is a header file

is an identifier

int is a keyword

main is an identifier

is an identifier

is an identifier

printf is an identifier

Hello have a great day is an argument

is an identifier

return is a keyword

0 is a number

is an identifier

Keywords: 2

Identifiers: 7

Operators: 0

Numbers: 1

The terminal window shows the output of the lexical analyzer. It starts with the command 'enter file path (drive:\fold\filename)' followed by 'sentence.txt'. The output then lists tokens and their types, followed by summary statistics.

```
enter file path (drive:\fold\filename)
sentence.txt
#include <stdio.h> is a header file
is an identifier
int is a keyword
main is an identifier
is an identifier
is an identifier
printf is an identifier
Hello have a great day is an argument
is an identifier
return is a keyword
0 is a number
is an identifier
Keywords: 2
Identifiers: 7
Operators: 0
Numbers: 1
```

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

The taskbar at the bottom shows various application icons, and the system tray indicates the date and time as 9:30 PM on 1/17/2022.

RESULT:

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

S. AKANSHA
RA1911029010060
P1

EXPERIMENT -2 CONVERSION FROM REGULAR EXPRESSION TO NFA

Aim:-

Conversion from 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

Procedure:-

The code for regular expression to NFA has been written in C language and run on a computer.

Code:-

```
#include<stdio.h>
#include <string.h>
int main()
{
char m[20],t[10][10];
int n, i , j , r=0, c=0;
printf("\n\t\t\tSIMULATION OF NFA");
printf("\n\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);

}
/*input: a+b*c */

```

Output:-

```
Enter a regular expression:a+b*c
```

```
0 1 2 3 4
```

```
      E      | 0  
E   E   E | 1  
      b    | 2  
      E   E | 3  
          | 4
```

```
Start state: 0
```

```
Final state: 4
```

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

Result:-

Thus the C program to convert regular expression to NFA has been executed and the output has been verified successfully.

EX. NO. 3 CONVERSION OF NFA TO DFA

S. Akansha
RA1911029010060

AIM:

To write a program for converting NFA to DFA.

ALGORITHM:

Start

Get the input from the user

Set the only state in SDFA to “unmarked”.

while SDFA contains an unmarked state do:

Let T be that unmarked state

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

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

Set MoveDFA(T,a) to S

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

Print the result.

Stop the program

PROGRAM:

```
import pandas as pd
```

```
# Taking NFA input from User
```

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

```

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

print("Enter final state of NFA : ")
nfa_final_state = [x for x in input().split()]    # Enter final state/states of NFA
#####
new_states_list = []           #holds all the new states created in dfa
dfa = {}                      #dfa dictionary/table or the output structure we
needed
keys_list = list(list(nfa.keys())[0])          #contains all the states in nfa plus the
states created in dfa are also appended further
path_list = list(nfa[keys_list[0]].keys())   #list of all the paths eg: [a,b] or [0,1]
#####
# Computing first row of DFA transition table

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

while len(new_states_list) != 0:          #condition is true only if the
new_states_list is not empty
    dfa[new_states_list[0]] = {}          #taking the first element of the
new_states_list and examining it
    for _ in range(len(new_states_list[0])):
        for i in range(len(path_list)):
            temp = []                  #creating a temporary list
            for j in range(len(new_states_list[0])):
                temp += nfa[new_states_list[0][j]][path_list[i]] #taking the union of the
states

```

```

s = ""
s = s.join(temp)           #creating a single string(new state) from all
the elements of the list
if s not in keys_list:
    new_states_list.append(s)
    keys_list.append(s)      #if the state is newly created
                           #then append it to the new_states_list
                           #as well as to the keys_list which contains
all the states
dfa[new_states_list[0]][path_list[i]] = s #assigning the new state in the DFA
table

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

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

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

print("\nFinal states of the DFA are : ",dfa_final_states)   #Printing Final states of
DFA

```

OUTPUT:

No. of states : 4

No. of transitions : 2

state name : A

path : a

Enter end state from state A travelling through path a :

A B

path : b

Enter end state from state A travelling through path b :

A

state name : B

path : a

Enter end state from state B travelling through path a :

C

path : b

Enter end state from state B travelling through path b :

C

state name : C

path : a

Enter end state from state C travelling through path a :

D

path : b

Enter end state from state C travelling through path b :

D

state name : D

path : a

Enter end state from state D travelling through path a :

path : b

Enter end state from state D travelling through path b :

NFA :-

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

Printing NFA table :-

a b

A [A, B] [A]

B [C] [C]

C [D] [D]

D [] []

Enter final state of NFA :

D

DFA :-

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

Printing DFA table :-

| | | |
|------|------|-----|
| | a | b |
| A | AB | A |
| AB | ABC | AC |
| ABC | ABCD | ACD |
| AC | ABD | AD |
| ABCD | ABCD | ACD |
| ACD | ABD | AD |
| ABD | ABC | AC |
| AD | AB | A |

Final states of the DFA are : ['ABCD', 'ACD', 'ABD', 'AD']

```
No. of states : 4
No. of transitions : 2
state name : A
path : a
Enter end state from state A travelling through path a :
A B
path : b
Enter end state from state A travelling through path b :
A
state name : B
path : a
Enter end state from state B travelling through path a :
C
path : b
Enter end state from state B travelling through path b :
C
state name : C
path : a
Enter end state from state C travelling through path a :
D
path : b
Enter end state from state C travelling through path b :
D
state name : D
path : a
Enter end state from state D travelling through path a :
path : b
Enter end state from state D travelling through path b :

NFA :-
{'A': {'a': ['A', 'B'], 'b': ['A']}, 'B': {'a': ['C'], 'b': ['C']}, 'C': {'a': ['D'], 'b': ['D']}, 'D': {'a': [], 'b': []}}
```

```

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

Printing NFA table :-
      a      b
A [A, B]  [A]
B   [C]    [C]
C   [D]    [D]
D   []     []

Enter final state of NFA :
D

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

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

Final states of the DFA are :  ['ABCD', 'ACD', 'ABD', 'AD']

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

```

RESULT :

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

S. AKANSHA
RA1911029010060 CSE-CN P1

EX. NO. 4(a)- ELIMINATION OF LEFT RECURSION

AIM:

A program for Elimination of Left Recursion.

ALGORITHM:

- Start the program.
- Initialize the arrays for taking input from the user.
- Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.
- Prompt the user to input the production for non-terminals.
- Eliminate left recursion using the following rules:- $A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_m$
 $A \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$
- Then replace it by
 - $A \rightarrow \beta_i A' \quad i=1,2,3,\dots, m$
 - $A' \rightarrow \alpha_j A' \quad j=1,2,3,\dots, n$
 - $A' \rightarrow \epsilon$
- After eliminating the left recursion by applying these rules, display the productions without left recursion.
- Stop.

PROGRAM:

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

int main()
{
int n;
cout<<"\nEnter number of non terminals: ";
cin>>n;
cout<<"\nEnter non terminals one by one: ";
int i;
vector<string> nonter(n);
vector<int> leftrecr(n,0);
for(i=0;i<n;++i) {
cout<<"\nNon terminal "<<i+1<<" : ";
cin>>nonter[i];
}
```

```

vector<vector<string>> prod;
cout<<"\nEnter '^' for null";
for(i=0;i<n;++i)
{
cout<<"\nNumber of "<<nonter[i]<<" productions: ";
int k;
cin>>k;
int j;
cout<<"\nOne by one enter all "<<nonter[i]<<" productions";
vector<string> temp(k);
for(j=0;j<k;++j)
{
cout<<"\nRHS of production "<<j+1<<": ";
string abc;
cin>>abc;
temp[j]=abc;
if(nonter[i].length()<=abc.length()&&nonter[i].compare(abc.substr(0,nonter[i].length())==0)
    leftrecr[i]=1;
}
prod.push_back(temp);
}
for(i=0;i<n;++i)
{
cout<<leftrecr[i];
}
for(i=0;i<n;++i)
{
if(leftrecr[i]==0)
continue;
int j;
nonter.push_back(nonter[i]+""");
vector<string> temp;
for(j=0;j<prod[i].size();++j)
{
if(nonter[i].length()<=prod[i][j].length()&&nonter[i].compare(prod[i][j].substr(0,nonter[i].length()))==0)
{
string abc=prod[i][j].substr(nonter[i].length(),prod[i][j].length()-nonter[i].length())+nonter[i]+""";
temp.push_back(abc);
prod[i].erase(prod[i].begin()+j);
--j;
}
}

```

```
else
{
prod[i][j]+=nonter[i]+""";
}
}
temp.push_back("^"); prod.push_back(temp);
}
cout<<"\n\n";
cout<<"\nNew set of non-terminals: ";
for(i=0;i<nonter.size();++i)
cout<<nonter[i]<<" ";
cout<<"\n\nNew set of productions: ";
for(i=0;i<nonter.size();++i)
{
int j; for(j=0;j<prod[i].size();++j)
{
cout<<"\n"<<nonter[i]<<" -> "<<prod[i][j];
}
}
return 0;
}
```

OUTPUT:

```
+_compiler
Enter number of non terminals: 2
Enter non terminals one by one:
Non terminal 1 : E
Non terminal 2 : T
Enter '^' for null
Number of E productions: 2
One by one enter all E productions
RHS of production 1: E+T
RHS of production 2: T
Number of T productions: 1
One by one enter all T productions
RHS of production 1: (E)
10

New set of non-terminals: E T E'
New set of productions:
E -> TE'
T -> (E)
E' -> +TE'
E' -> ^
...Program finished with exit code 0
```

RESULT:

A program for Elimination of Left Recursion was run successfully.

S.AKANSHA
RA1911029010060
CSE-CN P1

EX. NO. 4(b)-LEFT FACTORING

AIM :

A program for implementation Of Left Factoring

ALGORITHM :

- Start
- Ask the user to enter the set of productions.
- Check for common symbols in the given set of productions by comparing with:
 $A \rightarrow aB_1 | aB_2$
- If found, replace the particular productions with:
 $A \rightarrow aA' \quad A' \rightarrow B_1 | B_2 | \epsilon$
- Display the output.
- Exit

CODE :

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int n,j,l,i,m;
    int len[10] = {};
    string a, b1, b2, flag;
    char c;
    cout << "Enter the Parent Non-Terminal : ";
    cin >> c;
    a.push_back(c);
    b1 += a + "\'->";
    b2 += a + "\'\'->";;
    a += "->";
    cout << "Enter total number of productions : ";
    cin >> n;
    for (i = 0; i < n; i++)
    {
        cout << "Enter the Production " << i + 1 << " : ";
        cin >> flag;
        len[i] = flag.size();
        a += flag;
```

```

if (i != n - 1)
{
    a += "|";
}
cout << "The Production Rule is : " << a << endl;
char x = a[3];
for (i = 0, m = 3; i < n; i++)
{
    if (x != a[m])
    {
        while (a[m++] != '|');
    }
    else
    {
        if (a[m + 1] != '|')
        {
            b1 += "|" + a.substr(m + 1, len[i] - 1);
            a.erase(m - 1, len[i] + 1);
        }
        else
        {
            b1 += "#";
            a.insert(m + 1, 1, a[0]);
            a.insert(m + 2, 1, "\");
            m += 4;
        }
    }
}
char y = b1[6];
for (i = 0, m = 6; i < n - 1; i++)
{
    if (y == b1[m])
    {
        if (b1[m + 1] != '|')
        {
            flag.clear();
            for (int s = m + 1; s < b1.length(); s++)
            {
                flag.push_back(b1[s]);
            }
            b2 += "|" + flag;
            b1.erase(m - 1, flag.length() + 2);
        }
    }
}

```

```

        else
        {
            b1.insert(m + 1, 1, b1[0]);
            b1.insert(m + 2, 2, "\\");
            b2 += "#";
            m += 5;
        }
    }
b2.erase(b2.size() - 1);
cout << "After Left Factoring : " << endl;
cout << a << endl;
cout << b1 << endl;
cout << b2 << endl;
return 0;
}

```

OUTPUT :

| Run | Output |
|-----|--|
| | <pre> /tmp/kr05sGmzkS.o Enter the Parent Non-Terminal : M Enter total number of productions : 4 Enter the Production 1 : i Enter the Production 2 : iM Enter the Production 3 : (M) Enter the Production 4 : iM+M The Production Rule is : M->i iM (M) iM+M After Left Factoring : M->iM' (M) M' -># MM' M'' -># +M </pre> |

RESULT :

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

Ex-5: Computation of FIRST and FOLLOW

**T. AKANSHA
RA1911029010060
CSE-CN P1**

AIM:

To write a program for Computation of FIRST and FOLLOW.

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 FIRST_T 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.

PROGRAM:

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

int i,j,l,m,n=0,o,p,nv,z=0,x=0;
char str[10],temp,temp2[10],temp3[20],*ptr;

struct prod
{
    char lhs[10],rhs[10][10],ft[10],fol[10];
    int n;
}pro[10];

void findter()
{
    int k,t;
    for(k=0;k<n;k++)
    {
        if(temp==pro[k].lhs[0])
        {
            for(t=0;t<pro[k].n;t++)
            {
                if( pro[k].rhs[t][0]<65 || pro[k].rhs[t][0]>90 )
                    pro[i].ft[strlen(pro[i].ft)]=pro[k].rhs[t][0];
            }
        }
    }
}
```

```

        else if( pro[k].rhs[t][0]>=65 && pro[k].rhs[t][0]<=90 )
        {
            temp=pro[k].rhs[t][0];
            if(temp=='S')
                pro[i].ft[strlen(pro[i].ft)]='#';
            findter();
        }
    }
    break;
}
}

void findfol()
{
    int k,t,p1,o1,chk;
    char *ptr1;
    for(k=0;k<n;k++)
    {
        chk=0;
        for(t=0;t<pro[k].n;t++)
        {
            ptr1=strchr(pro[k].rhs[t],temp);
            if( ptr1 )
            {
                p1=ptr1-pro[k].rhs[t];
                if(pro[k].rhs[t][p1+1]>=65 && pro[k].rhs[t][p1+1]<=90)
                {
                    for(o1=0;o1<n;o1++)
                        if(pro[o1].lhs[0]==pro[k].rhs[t][p1+1])
                        {
                            strcat(pro[i].fol,pro[o1].ft);
                            chk++;
                        }
                }
                else if(pro[k].rhs[t][p1+1]=='\0')
                {
                    temp=pro[k].lhs[0];
                    if(pro[l].rhs[j][p]==temp)
                        continue;
                    if(temp=='S')
                        strcat(pro[i].fol,"$");
                    findfol();
                    chk++;
                }
            }
        }
        pro[i].fol[strlen(pro[i].fol)]=pro[k].rhs[t][p1+1];
        chk++;
    }
}
if(chk>0)
    break;
}

```

```

}

int main()
{
    FILE *f;
    //clrscr();

    for(i=0;i<10;i++)
        pro[i].n=0;

    f=fopen("tab5.txt","r");
    while(!feof(f))
    {
        fscanf(f,"%s",pro[n].lhs);
        if(n>0)
        {
            if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )
            {
                pro[n].lhs[0]='\0';
                fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);
                pro[n-1].n++;
                continue;
            }
        }
        fscanf(f,"%s",pro[n].rhs[pro[n].n]);
        pro[n].n++;
        n++;
    }

    printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");
    for(i=0;i<n;i++)
        for(j=0;j<pro[i].n;j++)
            printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);

    pro[0].ft[0]='#';
    for(i=0;i<n;i++)
    {
        for(j=0;j<pro[i].n;j++)
        {
            if( pro[i].rhs[j][0]<65 || pro[i].rhs[j][0]>90 )
            {
                pro[i].ft[strlen(pro[i].ft)]=pro[i].rhs[j][0];
            }
            else if( pro[i].rhs[j][0]>=65 && pro[i].rhs[j][0]<=90 )
            {
                temp=pro[i].rhs[j][0];
                if(temp=='S')
                    pro[i].ft[strlen(pro[i].ft)]='#';
                findter();
            }
        }
    }

    printf("\n\nFIRST\n");
    for(i=0;i<n;i++)

```

```

{
    printf("\n%s -> ",pro[i].lhs);
    for(j=0;j<strlen(pro[i].ft);j++)
    {
        for(l=j-1;l>=0;l--)
            if(pro[i].ft[l]==pro[i].ft[j])
                break;
        if(l===-1)
            printf("%c",pro[i].ft[j]);
    }
}

for(i=0;i<n;i++)
    temp2[i]=pro[i].lhs[0];
pro[0].fol[0]='$';
for(i=0;i<n;i++)
{
    for(l=0;l<n;l++)
    {
        for(j=0;j<pro[i].n;j++)
        {
            ptr=strchr(pro[l].rhs[j],temp2[i]);
            if( ptr )
            {
                p=ptr-pro[l].rhs[j];
                if(pro[l].rhs[j][p+1]>=65 && pro[l].rhs[j][p+1]<=90)
                {
                    for(o=0;o<n;o++)
                        if(pro[o].lhs[0]==pro[l].rhs[j][p+1])
                            strcat(pro[i].fol,pro[o].ft);
                }
                else if(pro[l].rhs[j][p+1]=='\0')
                {
                    temp=pro[l].lhs[0];
                    if(pro[l].rhs[j][p]==temp)
                        continue;
                    if(temp=='S')
                        strcat(pro[i].fol,"$");
                    findfol();
                }
                else
                    pro[i].fol[strlen(pro[i].fol)]=pro[l].rhs[j][p+1];
            }
        }
    }
}

printf("\n\nFOLLOW\n");
for(i=0;i<n;i++)
{
    printf("\n%s -> ",pro[i].lhs);
    for(j=0;j<strlen(pro[i].fol);j++)
    {
        for(l=j-1;l>=0;l--)
            if(pro[i].fol[l]==pro[i].fol[j])

```

```

        break;
    if(l== -1)
        printf("%c",pro[i].fol[j]);
    }
}
printf("\n");
//getch();
}

```

tab5.txt:

S ABE

S a

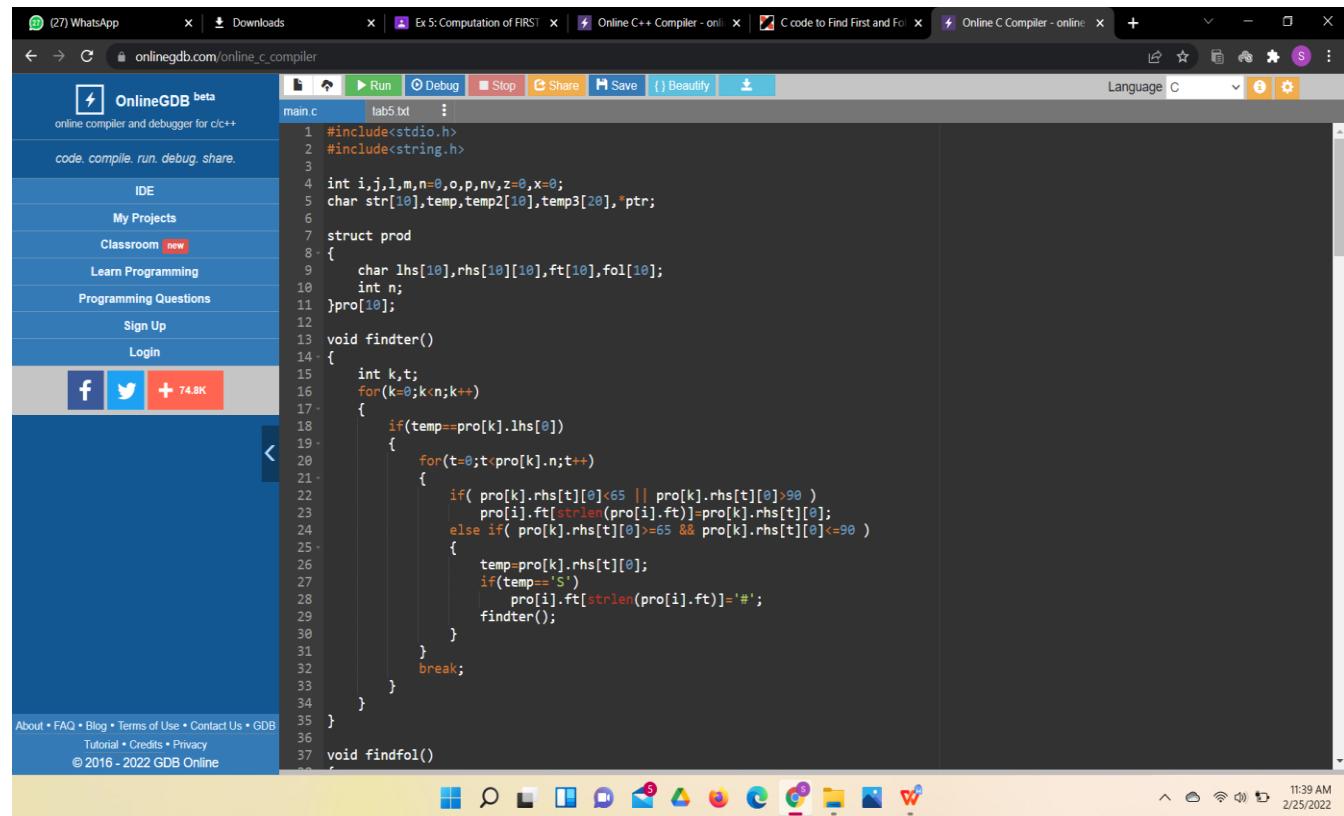
A p

A t

B Aq

S f

A w



The screenshot shows a web-based IDE interface for OnlineGDB. The code editor window displays a C program named 'main.c' with the following content:

```

1 #include<stdio.h>
2 #include<string.h>
3
4 int i,j,l,m,n,o,p,nv,z=0,x=0;
5 char str[10],temp,temp2[10],temp3[20],*ptr;
6
7 struct prod
8 {
9     char lhs[10],rhs[10][10],ft[10],fol[10];
10    int n;
11 }pro[10];
12
13 void findter()
14 {
15     int k,t;
16     for(k=0;k<n;k++)
17     {
18         if(temp==pro[k].lhs[0])
19         {
20             for(t=0;t<pro[k].n;t++)
21             {
22                 if( pro[k].rhs[t][0]<65 || pro[k].rhs[t][0]>90 )
23                     pro[i].ft[strlen(pro[i].ft)]=pro[k].rhs[t][0];
24                 else if( pro[k].rhs[t][0]>=65 && pro[k].rhs[t][0]<=90 )
25                 {
26                     temp=pro[k].rhs[t][0];
27                     if(temp=='$')
28                         pro[i].ft[strlen(pro[i].ft)]='#';
29                     finder();
30                 }
31             }
32             break;
33         }
34     }
35 }
36
37 void findfol()
38 {

```

The browser's address bar shows the URL `onlinegdb.com/online_c_compiler`. The status bar at the bottom right indicates the date and time as `2/25/2022 11:39 AM`.

The screenshot shows the OnlineGDB beta interface. The left sidebar has sections for IDE, My Projects, Classroom (new), Learn Programming, Programming Questions, Sign Up, and Login. It also features social sharing icons for Facebook, Twitter, and LinkedIn, with a count of 74.8K.

The main workspace displays the following C code:

```
ptr1=strchr(pro[k].rhs[t],temp);
if( ptr1 )
{
    p1=ptr1-pro[k].rhs[t];
    if(pro[k].rhs[t][p1+1]>=65 && pro[k].rhs[t][p1+1]<=90)
    {
        for(o1=0;o1<n;o1++)
            if(pro[o1].lhs[0]==pro[k].rhs[t][p1+1])
            {
                strcat(pro[i].fol,pro[o1].ft);
                chk++;
            }
    }
    else if(pro[k].rhs[t][p1+1]=='\0')
    {
        temp=pro[k].lhs[0];
        if(pro[l].rhs[j][p]==temp)
            continue;
        if(temp=='$')
            strcat(pro[i].fol,"$");
        findfol();
        chk++;
    }
    else
    {
        pro[i].fol[strlen(pro[i].fol)]=pro[k].rhs[t][p1+1];
        chk++;
    }
}
if(chk>0)
break;
}

int main()
{
```

The status bar at the bottom right indicates the date as 2/25/2022 and the time as 11:39 AM.

This screenshot shows the same OnlineGDB interface as the previous one, but with different code in the editor.

The main workspace displays the following C code:

```
FILE *f;
//clrscr();
for(i=0;i<10;i++)
    pro[i].n=0;

f=fopen("tab5.txt","r");
while(!feof(f))
{
    fscanf(f,"%s",pro[n].lhs);
    if(n>0)
    {
        if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )
        {
            pro[n].lhs[0]='\0';
            fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);
            pro[n-1].n++;
            continue;
        }
    }
    fscanf(f,"%s",pro[n].rhs[pro[n].n]);
    pro[n].n++;
    n++;
}

printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");
for(i=0;i<n;i++)
    for(j=0;j<pro[i].n;j++)
        printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);

pro[0].ft[0]='#';
for(i=0;i<n;i++)
{
    for(j=0;j<pro[i].n;j++)
    {
        if( pro[i].rhs[j][0]<65 || pro[i].rhs[j][0]>90 )
```

The status bar at the bottom right indicates the date as 2/25/2022 and the time as 11:39 AM.

The screenshot shows the OnlineGDB beta IDE interface. The left sidebar contains links for IDE, My Projects, Classroom, Learn Programming, Programming Questions, Sign Up, and Login, along with social media sharing buttons for Facebook, Twitter, and LinkedIn.

The main workspace displays a C program named 'main.c' with line numbers 143 to 179. The code implements a function to compute FIRST sets for a grammar. It uses arrays 'pro' and 'temp2' to store non-terminal symbols and their right-hand sides. The algorithm iterates through non-terminals and their right-hand sides, comparing them with other non-terminals to find common prefixes.

```
main.c
143     }
144     }
145
146     for(i=0;i<n;i++)
147         temp2[i]=pro[i].lhs[0];
148     pro[0].fol[0]='$';
149     for(i=0;i<n;i++)
150     {
151         for(l=0;l<n;l++)
152         {
153             for(j=0;j<pro[i].n;j++)
154             {
155                 ptr=strchr(pro[l].rhs[j],temp2[i]);
156                 if( ptr )
157                 {
158                     p=ptr-pro[l].rhs[j];
159                     if(pro[l].rhs[j][p+1]>=65 && pro[l].rhs[j][p+1]<=90)
160                     {
161                         for(o=0;o<n;o++)
162                             if(pro[o].lhs[0]==pro[l].rhs[j][p+1])
163                                 strcat(pro[i].fol,pro[o].ft);
164                     }
165                     else if(pro[l].rhs[j][p+1]=='\0')
166                     {
167                         temp=pro[l].lhs[0];
168                         if(pro[l].rhs[j][p]==temp)
169                             continue;
170                         if(temp=='$')
171                             strcat(pro[i].fol,"$");
172                         findfol();
173                     }
174                 }
175                 pro[i].fol[strlen(pro[i].fol)]=pro[l].rhs[j][p+1];
176             }
177         }
178     }
179 }
```

The bottom status bar shows the date and time: 2/25/2022 11:39 AM.

This screenshot shows the same OnlineGDB IDE interface after running the provided C code. The workspace now displays the output of the computation, which consists of a list of strings:

```
1 S ABE
2 S a
3 A p
4 A t
5 B Aq
6 S f
7 A w
```

OUTPUT:

The screenshot shows a browser window with multiple tabs open. The active tab is 'Ex 5: Computation of FIRST' on onlinegdb.com. The page displays a grammar and its FIRST and FOLLOW sets.

THE GRAMMAR IS AS FOLLOWS

```
S -> ABE  
S -> a  
A -> p  
A -> t  
B -> Aq  
S -> f  
A -> w
```

FIRST

```
S -> #pta  
A -> pt  
B -> pt  
S -> f  
A -> w
```

FOLLOW

```
S -> $  
A -> ptq  
B ->  
S ->  
A -> ptq
```

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

RESULT:

The computation of First and Follow in C was compiled, executed and verified successfully.

S. AKANSHA
RA1911029010060 CSE-CN P1

EXPERIMENT-6 CONSTRUCTION OF PREDICTIVE PARSING

AIM:

A program to construct Predictive Parsing.

ALGORITHM:

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

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

CODE:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
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;
            11:if(!((st[i][j]>64)&&(st[i][j]<91)))
            {
```

```

for(m=0;m<l;m++)
{
    if(ft[i][m]==st[i][j])
        goto s1;
    }
    ft[i][l]=st[i][j];
    l=l+1;
    s1:j=j+1;
}
else
{
    if(s>0)
    {
        while(st[i][j]!=st[a][0])
        {
            a++;
        }
        b=0;
        while(ft[a][b]!='\0')
        {
            for(m=0;m<l;m++)
            {
                if(ft[i][m]==ft[a][b])
                    goto s2;
                }
                ft[i][l]=ft[a][b];
                l=l+1;
                s2:b=b+1;
            }
        }
    }
    while(st[i][j]!='\0')
    {
        if(st[i][j]=='|')
        {
            j=j+1;
            goto l1;
        }
        j=j+1;
    }
    ft[i][l]='\0';
}
printf("first \n");
for(i=0;i<n;i++)
    printf("FIRS[%c]=%s\n",st[i][0],ft[i]);
fol[0][0]='$';
for(i=0;i<n;i++)
{

```

```

k=0;
j=3;
if(i==0)
    l=1;
else
    l=0;
k1:while((st[i][0]!=st[k][j])&&(k<n))
{
    if(st[k][j]=='\0')
    {
        k++;
        j=2;
    }
    j++;
}

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

```

```

        l=l+1;
    }
    else
    e=1;
    q2:p++;
}
if(e==1)
{
    e=0;
    goto a1;
}
}
else
{
    a1:c=0;
    a=0;
    while(st[k][0]!=st[a][0])
    {
        a++;
    }
    while((fol[a][c]!='0')&&(st[a][0]!=st[i][0]))
    {
        for(m=0;m<l;m++)
        {
            if(fol[i][m]==fol[a][c])
            goto q1;
        }
        fol[i][l]=fol[a][c];
        l++;
        q1:c++;
    }
    goto k1;
}
fol[i][l]='\0';
}
printf("follow \n");
for(i=0;i<n;i++)
    printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);
printf("\n");
s=0;
for(i=0;i<n;i++)
{
    j=3;
    while(st[i][j]!='0')
    {
        if((st[i][j-1]=='|')||(j==3))
        {
            for(p=0;p<=2;p++)

```

```

{
    fin[s][p]=st[i][p];
}
t=j;
for(p=3;((st[i][j]!='')&&(st[i][j]!='0'));p++)
{
    fin[s][p]=st[i][j];
    j++;
}
fin[s][p]='\0';
if(st[i][k]=='@')
{
    b=0;
    a=0;
    while(st[a][0]!=st[i][0])
    {
        a++;
    }
    while(fol[a][b]!='0')
    {
        printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],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++;
}
}

```

Screenshot of the OnlineGDB IDE showing a C program for LR(0) parser construction.

```
main.c
1 #include<stdio.h>
2 #include<conio.h>
3 #include<string.h>
4 void main()
5 {
6     char fin[10][20],st[10][20],ft[20][20],fol[20][20];
7     int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;
8
9     printf("enter the no. of nonterminals\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            }
32            while(st[i][j]!='0')
33            {
34                a++;
35            }
36            b=0;
37            while(ft[a][b]!='0')
38            {
39                for(m=0;m<l;m++)
40                {
41                    if(ft[i][m]==ft[a][b])
42                        goto s2;
43                }
44                ft[i][l]=ft[a][b];
45                l=l+1;
46                s2:b=b+1;
47            }
48        }
49    }
50    while(st[i][j]!='0')
51    {
52        if(st[i][j]=='|')
53        {
54            j=j+1;
55            goto l1;
56        }
57        j=j+1;
58    }
59    ft[i][l]='\0';
60 }
```

The code implements an LR(0) parser construction algorithm. It reads the grammar (non-terminals and productions) and builds an NFA (ft) and DFA (st). The NFA transitions are based on terminals and non-terminals. The DFA transitions are based on non-terminals. The parser then derives the start symbol (ft[i][l]) from the DFA states.

Screenshot of the OnlineGDB IDE showing the same C program for LR(0) parser construction.

```
main.c
1 #include<stdio.h>
2 #include<conio.h>
3 #include<string.h>
4 void main()
5 {
6     char fin[10][20],st[10][20],ft[20][20],fol[20][20];
7     int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;
8
9     printf("enter the no. of nonterminals\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            }
32            while(st[i][j]!='0')
33            {
34                a++;
35            }
36            b=0;
37            while(ft[a][b]!='0')
38            {
39                for(m=0;m<l;m++)
40                {
41                    if(ft[i][m]==ft[a][b])
42                        goto s2;
43                }
44                ft[i][l]=ft[a][b];
45                l=l+1;
46                s2:b=b+1;
47            }
48        }
49    }
50    while(st[i][j]!='0')
51    {
52        if(st[i][j]=='|')
53        {
54            j=j+1;
55            goto l1;
56        }
57        j=j+1;
58    }
59    ft[i][l]='\0';
60 }
```

The code is identical to the one in the first screenshot, implementing the LR(0) parser construction algorithm.

The screenshot shows the OnlineGDB IDE interface. On the left is a sidebar with links for 'My Projects', 'Classroom', 'Learn Programming', and 'Programming Questions'. The main area has tabs for 'main.c' and 'input'. The code editor displays the following C code:

```
main.c
76    i=1;
77    else
78    l=0;
79    k1:while((st[i][@]!=st[k][j])&&(k<n))
80    {
81        if(st[k][j]=='\0')
82        {
83            k++;
84            j=2;
85        }
86        j++;
87    }
88
89    j=j+1;
90    if(st[i][@]==st[k][j-1])
91    {
92        if((st[k][j]!='\1')&&(st[k][j]!='\0'))
93        {
94            a=0;
95            if(((st[k][j]>64)&&(st[k][j]<91)))
96            {
97                for(m=0;m<l;m++)
98                {
99                    if(fol[i][m]==st[k][j])
100                   goto q3;
101                fol[i][l]=st[k][j];
102                l++;
103            }
104            q3:;
105        }
106    }
107
108
M[C,d]=C->d
```

The output window at the bottom shows the message: "...Program finished with exit code 0 Press ENTER to exit console.[]". The status bar indicates the date and time: 3/4/2022 12:17 PM.

This screenshot shows the continuation of the C program from the previous one. The code editor now displays lines 119 through 149 of the 'main.c' file. The output window shows the same completion message: "...Program finished with exit code 0 Press ENTER to exit console.[]". The status bar indicates the date and time: 3/4/2022 12:17 PM.

The screenshot shows the OnlineGDB beta IDE interface. The main window displays a C program named 'main.c' with line numbers 156 to 385. The code implements a predictive parser for a grammar. It includes functions for printing follow sets and tokens, and a main loop for processing input. The status bar at the bottom shows '...Program finished with exit code 0' and 'Press ENTER to exit console.' The taskbar at the bottom of the screen shows various application icons.

```
156     }
157     }
158     }
159     }
160     fol[i][1]='\0';
161     }
162     printf("follow \n");
163     for(i=0;i<n;i++)
164     {
165         printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);
166         s=0;
167         for(i=0;i<n;i++)
168         {
169             j=3;
170             while(st[i][j]!='\0')
171             {
172                 if((st[i][j-1]=='|')||(j==3))
173                 {
174                     for(p=0;p<=2;p++)
175                     {
176                         fin[s][p]=st[i][p];
177                     }
178                     t=j;
179                     for(p=3;((st[i][j]=='|')&&(st[i][j]=='\0'));p++)
180                     {
181                         fin[s][p]=st[i][j];
182                         j++;
183                     }
184                     fin[s][p]='\0';
185                     if(st[i][k]=='@')
186                 }
187             }
188         }
189     }
190     while(fol[a][b]!='\0')
191     {
192         printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);
193         b++;
194     }
195     else if(!((st[i][t]>64)&&(st[i][t]<91)))
196     {
197         printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);
198     }
199     else
200     {
201         b=0;
202         a=0;
203         while(st[a][0]!=st[i][3])
204         {
205             a++;
206         }
207         while(st[a][b]!='\0')
208         {
209             printf("M[%c,%c]=%s\n",st[i][0],st[a][b],fin[s]);
210             b++;
211         }
212     }
213     s++;
214     }
215     if(st[i][j]=='|')
216     {
217         j++;
218     }
219 }
220 }
221 }
222 }
```

This screenshot shows the same OnlineGDB IDE interface, but with a different version of the 'main.c' program. The code has been modified to handle terminals with ASCII values greater than 64. The changes include additional logic in the main loop to correctly identify terminals like 'A', 'B', etc., which have ASCII values like 65 and 66 respectively. The rest of the parser logic remains the same. The status bar and taskbar are identical to the first screenshot.

```
156     }
157     }
158     }
159     }
160     fol[i][1]='\0';
161     }
162     printf("follow \n");
163     for(i=0;i<n;i++)
164     {
165         printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);
166         s=0;
167         for(i=0;i<n;i++)
168         {
169             j=3;
170             while(st[i][j]!='\0')
171             {
172                 if((st[i][j-1]=='|')||(j==3))
173                 {
174                     for(p=0;p<=2;p++)
175                     {
176                         fin[s][p]=st[i][p];
177                     }
178                     t=j;
179                     for(p=3;((st[i][j]=='|')&&(st[i][j]=='\0'));p++)
180                     {
181                         fin[s][p]=st[i][j];
182                         j++;
183                     }
184                     fin[s][p]='\0';
185                     if(st[i][k]=='@')
186                 }
187             }
188         }
189     }
190     while(fol[a][b]!='\0')
191     {
192         printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);
193         b++;
194     }
195     else if(!((st[i][t]>64)&&(st[i][t]<91)))
196     {
197         printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);
198     }
199     else
200     {
201         b=0;
202         a=0;
203         while(st[a][0]!=st[i][3])
204         {
205             a++;
206         }
207         while(st[a][b]!='\0')
208         {
209             printf("M[%c,%c]=%s\n",st[i][0],st[a][b],fin[s]);
210             b++;
211         }
212     }
213     s++;
214     }
215     if(st[i][j]=='|')
216     {
217         j++;
218     }
219 }
220 }
221 }
222 }
```

OUTPUT:

Enter the no. of nonterminals

2

Enter the productions in a grammar

S->CC

C->eC | d

first

FIRS[S] = ed

FIRS[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

The screenshot shows a window titled "Ex 6: Construction of Predictive P." with tabs for "Online C Compiler - online editor" and a "+" button. The main area is labeled ".c_compiler". It has a toolbar with icons for file operations (New, Open, Save, Print, Run, Stop, Share, Beautify) and a download button. Below the toolbar, there are two tabs: "main.c" (selected) and "input". The "input" tab contains the user input for the grammar construction. The "main.c" tab shows the generated C code for the predictive parser. The terminal window at the bottom displays the execution results, including the user input and the generated first and follow sets.

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

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

RESULT:

The program was successfully compiled and executed.

EXPERIMENT-7 Implementation of Shift Reduce Parsing

AIM:

A program to implement shift reduce parsing.

ALGORITHM:

1. Get the input expression and store it in the input buffer.
2. Read the data from the input buffer one at the time.
3. Using stack and push & pop operation shift and reduce symbols with respect to production rules available.
4. Continue the process till symbol shift and production rule reduce reaches the start symbol.
5. Display the Stack Implementation table with corresponding Stack actions with input symbols.

CODE:

```
#include<stdio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
int main()
{
    puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
    puts("enter input string ");
    gets(a);
    c=strlen(a);
    strcpy(act,"SHIFT->");
    puts("stack \t input \t action");
    for(k=0,i=0; j<c; k++,i++,j++)
    {
        if(a[j]=='i' && a[j+1]=='d')
        {
            stk[i]=a[j];
            stk[i+1]=a[j+1];
            stk[i+2]='\0';
            a[j]=' ';
            a[j+1]=' ';
            printf("\n$%s\t%s$\t%s",stk,a,act);
            check();
        }
        else
        {
            stk[i]=a[j];
            stk[i+1]='\0';
            a[j]=' ';
            printf("\n$%s\t%s$\t%s",stk,a,act);
        }
    }
}
```

```

        check();
    }
}

void check()
{
    strcpy(ac,"REDUCE TO E");
    for(z=0; z<c; z++)
        if(stk[z]=='i' && stk[z+1]=='d')
    {
        stk[z]='E';
        stk[z+1]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        j++;
    }
    for(z=0; z<c; z++)
        if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
    {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        i=i-2;
    }
    for(z=0; z<c; z++)
        if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
    {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        i=i-2;
    }
    for(z=0; z<c; z++)
        if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
    {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        i=i-2;
    }
}

```

Meet - fo-h-xei-ost | Ex 7: Implementation of Shift Reduc... | Compiler Design : Shift Reducing | Online C Compiler - online editor

onlinegdb.com/online_c_compiler

OnlineGDB beta
online compiler and debugger for c/c++
code. compile. run. debug. share.

IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Sign Up
Login

76.1K

main.c

```
1 #include<stdio.h>
2 #include<string.h>
3 int k=0,z=0,i=0,j=0,c=0;
4 char a[16],ac[20],stk[15],act[10];
5 void check();
6 int main()
7 {
8     puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
9     puts("enter input string ");
10    gets(a);
11    c=strlen(a);
12    strcpy(act,"SHIFT->");
13    puts("stack \t input \t action");
14    for(k=0,i=0; j<c; k++,i++,j++)
15    {
16        if(a[j]=='i' && a[j+1]=='d')
17        {
18            stk[i]=a[j];
19            stk[i+1]=a[j+1];
20            stk[i+2]='\0';
21            a[j]=' ';
22            a[j+1]=' ';
23            printf("\n$%s\t%s$\t%s",stk,a,act);
24            check();
25        }
26        else
27        {
28            stk[i]=a[j];
29            stk[i+1]='\0';
30            a[j]=' ';
31            printf("\n$%s\t%s$\t%s",stk,a,act);
32            check();
33        }
34    }
35 }
36
37 void check()
38 {
39     strcpy(ac,"REDUCE TO E");
40     for(z=0; z<c; z++)
41         if(stk[z]=='i' && stk[z+1]=='d')
42         {
43             stk[z]='E';
44             stk[z+1]='\0';
45             printf("\n$%s\t%s$\t%s",stk,a,ac);
46             j++;
47         }
48     for(z=0; z<c; z++)
49         if(stk[z]=='E' && stk[z+1]==',' && stk[z+2]=='E')
50         {
51             stk[z]='E';
52             stk[z+1]='\0';
53             stk[z+2]='\0';
54             printf("\n$%s\t%s$\t%s",stk,a,ac);
55             i=i-2;
56         }
57     for(z=0; z<c; z++)
58         if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
59         {
60             stk[z]='E';
61             stk[z+1]='\0';
62             stk[z+2]='\0';
63             printf("\n$%s\t%s$\t%s",stk,a,ac);
64             i=i-2;
65         }
66     for(z=0; z<c; z++)
67 }
```

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

About • FAQ • Blog • Terms of Use • Contact Us • GDB
Tutorial • Credits • Privacy
© 2016 - 2022 GDB Online

88°F Haze

11:25 AM 3/11/2022

Meet - fo-h-xei-ost | Ex 7: Implementation of Shift Reduc... | Compiler Design : Shift Reducing | Online C Compiler - online editor

onlinegdb.com/online_c_compiler

OnlineGDB beta
online compiler and debugger for c/c++
code. compile. run. debug. share.

IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Sign Up
Login

76.1K

main.c

```
1 }
2 }
3 }
4 }
5 }
6 }
7 }
8 }
9 }
10 }
11 }
12 }
13 }
14 }
15 }
16 }
17 }
18 }
19 }
20 }
21 }
22 }
23 }
24 }
25 }
26 }
27 }
28 }
29 }
30 }
31 }
32 }
33 }
34 }
35 }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
```

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

About • FAQ • Blog • Terms of Use • Contact Us • GDB
Tutorial • Credits • Privacy
© 2016 - 2022 GDB Online

88°F Haze

11:25 AM 3/11/2022

The screenshot shows the OnlineGDB beta interface. On the left, there's a sidebar with links for 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Sign Up', and 'Login'. Below the sidebar are social sharing icons for Facebook, Twitter, and LinkedIn, followed by a '76.1K' count.

The main area has tabs for 'main.c' and 'Output'. The code in 'main.c' is a C program that implements a shift-reduce parser. It uses a stack (stk) to store tokens and actions (a, ac). The program processes input tokens and performs shifts (adding tokens to the stack) or reduces (replacing tokens with symbols from the grammar).

```

main.c
44     stk[z]='E';
45     stk[z+1]='\0';
46     printf("\n$%s\t%s$\t%s",stk,a,ac);
47     j++;
48 }
49 for(z=0; z<c; z++)
50     if(stk[z]=='E' && stk[z+1]=='+') && stk[z+2]=='E')
51     {
52         stk[z]='E';
53         stk[z+1]='\0';
54         stk[z+2]='\0';
55         printf("\n$%s\t%s$\t%s",stk,a,ac);
56         i=i-2;
57     }
58 for(z=0; z<c; z++)
59     if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
60     {
61         stk[z]='E';
62         stk[z+1]='\0';
63         stk[z+2]='\0';
64         printf("\n$%s\t%s$\t%s",stk,a,ac);
65         i=i-2;
66     }
67 for(z=0; z<c; z++)
68     if(stk[z]=='(' && stk[z+1]==')' && stk[z+2]==')')
69     {
70         stk[z]='E';
71         stk[z+1]='\0';
72         stk[z+2]='\0';
73         printf("\n$%s\t%s$\t%s",stk,a,ac);
74         i=i-2;
75     }
76 }

```

The 'Output' tab shows the result of running the program:

```

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

```

OUTPUT:

GRAMMAR is E->E+E

E->E*E

E->(E)

E->id

enter input string

id+id*id+id

stack input action

| | | |
|--------|-------------|----------------|
| \$id | +id*id+id\$ | SHIFT->id |
| \$E | +id*id+id\$ | REDUCE TO E |
| \$E+ | id*id+id\$ | SHIFT->symbols |
| \$E+id | *id+id\$ | SHIFT->id |
| \$E+E | *id+id\$ | REDUCE TO E |
| \$E | *id+id\$ | REDUCE TO E |
| \$E* | id+id\$ | SHIFT->symbols |
| \$E*id | +id\$ | SHIFT->id |
| \$E*E | +id\$ | REDUCE TO E |
| \$E | +id\$ | REDUCE TO E |
| \$E+ | id\$ | SHIFT->symbols |
| \$E+id | \$ | SHIFT->id |
| \$E+E | \$ | REDUCE TO E |
| \$E | \$ | REDUCE TO E |

```
GRAMMAR is E->E+E
E->E*E
E->(E)
E->id
enter input string
id+id*id+id
stack      input    action
<
$Id      +id*id+id$    SHIFT->id
$E       +id*id+id$    REDUCE TO E
$E+      id*id+id$    SHIFT->symbols
$E+id    *id+id$      SHIFT->id
$E+E    *id+id$      REDUCE TO E
$E       *id+id$      REDUCE TO E
$E*      id+id$      SHIFT->symbols
$E+id   +id$        SHIFT->id
$E+E   +id$        REDUCE TO E
$E       +id$        REDUCE TO E
$E+      id$        SHIFT->symbols
$E+id   $          SHIFT->id
$E+E   $          REDUCE TO E
$E       $          REDUCE TO E
GDB ...Program finished with exit code 0
Press ENTER to exit console.
```



RESULT:

The program was successfully compiled and executed.

S. AKANSHA

RA191102901006

0 CSE-CN P1

EXP 8: Construction of Leading and Trailing

Aim:

To write a program for implementing 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

Main.c:

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

int vars,terms,i,j,k,m,rep,count,temp=-1;
char var[10],term[10],lead[10][10],trail[10][10];
struct grammar
{
    int prodno;
```

```

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

```

```

{
for(i=0;i<vars;i++)
{
    for(j=0;j<gram[i].prodno;j++)
    {
        for(k=0;k<terms;k++)
        {
            if(gram[i].rhs[j][0]==term[k])
                lead[i][k]=1;
            else
            {
                if(gram[i].rhs[j][1]==term[k])
                    lead[i][k]=1;
            }
        }
    }
}

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

```

```

        out:
        for(k=0;k<terms;k++)
        {
            if(lead[temp][k]==1)
                lead[i][k]=1;
        }
    }

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

```

```

for(rep=0;rep<vars;rep++)
{
    for(i=0;i<vars;i++)
    {
        for(j=0;j<gram[i].prodno;j++)
        {
            count=0;
            while(gram[i].rhs[j][count]!='x0')
                count++;
            for(m=1;m<vars;m++)
            {
                if(gram[i].rhs[j][count-1]==var[m])
                    temp=m;
            }
            for(k=0;k<terms;k++)
            {
                if(trail[temp][k]==1)
                    trail[i][k]=1;
            }
        }
    }
}

void display()
{
    for(i=0;i<vars;i++)
    {
        cout<<"\nLEADING("<<gram[i].lhs<<") = ";
        for(j=0;j<terms;j++)
        {
            if(lead[i][j]==1)

```

```

cout<<term[j]<<",";
}
}

cout<<endl;
for(i=0;i<var
s;i++)
{
cout<<"\nTRAILING("<<gram[i].lhs<<""
)= "; for(j=0;j<terms;j++)
{
if(trail[i][j]==1)
cout<<term[j]<<",";
}
}

void main()
{
clrscr();
get();
leading();
trailing();
display();
getch();
}

```

Output:

```
----- LEADING AND TRAILING -----  
Enter the no. of variables : 3  
Enter the variables :  
E  
T  
F  
Enter the no. of terminals : 5  
Enter the terminals : )  
(  
*  
+  
i  
----- PRODUCTION DETAILS -----  
Enter the no. of production of E:2  
E->E+T  
E->T  
Enter the no. of production of T:2  
T->T*i  
T->F  
Enter the no. of production of F:2  
F->(E)  
F->i  
LEADING(E) = (,*,+,,i,  
LEADING(T) = (,*,,i,  
LEADING(F) = (,,i,  
TRAILING(E) = ),*,+,i,  
TRAILING(T) = ),*,i,  
TRAILING(F) = ),i,
```

Result:

The program to implementing leading and trailing was completed successfully.

S. AKANSHA
RA1911029010060 CSE CN P1

EXP 9 : Computation of LR(0) Items

Aim:

A program to implement LR(0) items

Algorithm:-

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

Program:

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

char prod[20][20],listofvar[26]={"ABCDEFGHIJKLMNPQR"};
int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
```

```

int noitem=0;

struct Grammar

{
    char lhs; char rhs[8];

}g[20],item[20],clos[20][10];



int isvariable(char variable)
{ for(int i=0;i<novar;i++)
    if(g[i].lhs==variable) return
    i+1;
    return 0;
}

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

```

```

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

    if(l==n)
    {
        clos[noitem][n].lhs=clos[0][k].lhs;
        strcpy(clos[noitem][n].rhs,clos[0][k].rhs);
        n=n+1;

    }
}
}

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

```

```

        c=c+1;
        if(c==arr[i])
        { flag=1; goto
          exit;
        }
      }

}

exit:; if(flag==0)
arr[noitem++]=n;

}

```

```

void main()
{
  clrscr();
  cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n";
  do
  { cin>>prod[i++];
  }while(strcmp(prod[i-1],"0")!=0); for(n=0;n<i-
  1;n++)
  {
    m=0;
    j=novar;
    g[novar++].lhs=prod[n][0];
    for(k=3;k<strlen(prod[n]);k++)
    {
      if(prod[n][k] != '|')
        g[j].rhs[m++]=prod[n][k];
      if(prod[n][k]== '|')
      {
        g[j].rhs[m]='\0';

```

```

m=0;
j=novar;
g[novar++].lhs=prod[n][0];

}

} } for(i=0;i<26;i++)
if(!isvariable(listofvar[i])) break;
g[0].lhs=listofvar[i]; char temp[2]={g[1].lhs,'0'};
strcat(g[0].rhs,temp); cout<<"\n\n augmented
grammar \n"; for(i=0;i<novar;i++)
cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";
getch();

for(i=0;i<novar;i++)
{
    clos[noitem][i].lhs=g[i].lhs;
    strcpy(clos[noitem][i].rhs,g[i].rhs);
    if(strcmp(clos[noitem][i].rhs,"ε")==0)
        strcpy(clos[noitem][i].rhs,".");
    else
    {
        for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)
            clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1];
        clos[noitem][i].rhs[0]='.';
    }
}
arr[noitem++]=novar;
for(int z=0;z<noitem;z++)
{
    char list[10]; int
    l=0;
    for(j=0;j<arr[z];j++)
    )
    {

```

```

for(k=0;k<strlen(clos[z][j].rhs)-1;k++)
{
    if(clos[z][j].rhs[k]=='.')
    { for(m=0;m<l;m++)
        if(list[m]==clos[z][j].rhs[k+1])
            break;
        if(m==l)
            list[l++]=clos[z][j].rhs[k+1];
    }
} } for(int
x=0;x<l;x++)
findclosure(z,list[x]);
}

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

```

Output:

```
ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :  
E->E+T  
E->T  
T->T*T  
T->F  
F->( E )  
F->i  
0  
  
augumented grammar  
  
A->E  
E->E+T  
E->T  
T->T*T  
T->F  
F->( E )  
F->i  
THE SET OF ITEMS ARE  
  
10  
  
A->.E  
E->..E+T  
E->..T  
T->..T*T  
T->..F  
F->..( E )  
F->..i
```

```
I1
A->E .
E->E . + T

I2
E->T .
T->T . * F

I3
T->F .

I4
F->{ . E )
E-> . E + T
E-> . T
T-> . T * F
T-> . F
F-> . ( E )
F-> . i

I5
F->i .

I6
E->E + . T
```

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

I7
T->T * . F
F-> . ( E )
F-> . i

I8
F->{ E . )
E->E . + T

I9
E->E + T .
T->T . * F

I10
T->T * F .

I11
F->( E ) .

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

Result:

The program was successfully compiled and run.

EXP 10: Intermediate code generation – Postfix, Prefix

S. AKANSHA
RA1911029010060 CSE-CN P1

Aim:

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

Algorithm:

- Declare set of operators.
- Initialize an empty stack.
- To convert INFIX to POSTFIX follow the following steps
- Scan the infix expression from left to right.
- If the scanned character is an operand, output it.
- 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.
- 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.
- If the scanned character is an ‘(‘ , push it to the stack.
- If the scanned character is an ‘)’ , pop the stack and output it until a ‘(‘ is encountered, and discard both the parenthesis.
- Pop and output from the stack until it is not empty.
- To convert INFIX to PREFIX follow the following steps
- First, reverse the infix expression given in the problem.
- Scan the expression from left to right.
- Whenever the operands arrive, print them.
- If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
- Repeat steps 6 to 9 until the stack is empty.

Code:

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

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

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

```
def infix_to_postfix(formula):
```

```
    stack = [] # only pop when the coming op has priority
```

```
    output = "
```

```
    for ch in formula:
```

```
        if ch not in OPERATORS:
```

```
            output += ch
```

```
        elif ch == '(':
```

```
            stack.append('(')
```

```
        elif ch == ')':
```

```
            while stack and stack[-1] != '(':
```

```
                output += stack.pop()
```

```
            stack.pop() # pop '('
```

```
        else:
```

```
            while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
```

```
                output += stack.pop()
```

```
            stack.append(ch)
```

```
# leftover
```

```
while stack:
```

```
    output += stack.pop()
```

```
print(f'POSTFIX: {output}')
```

```
    return output
```

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

```
def infix_to_prefix(formula):
    op_stack = []
    exp_stack = []

    for ch in formula:
        if not ch in OPERATORS:
            exp_stack.append(ch)
        elif ch == '(':
            op_stack.append(ch)
        elif ch == ')':
            while op_stack[-1] != '(':
                op = op_stack.pop()
                a = exp_stack.pop()
                b = exp_stack.pop()
                exp_stack.append(op + b + a)
            op_stack.pop() # pop '('
        else:
            while op_stack and op_stack[-1] != '(' and PRI[ch] <= PRI[op_stack[-1]]:
                op = op_stack.pop()
                a = exp_stack.pop()
                b = exp_stack.pop()
```

```

exp_stack.append(op + b + a)

op_stack.append(ch)

# leftover

while op_stack:
    op = op_stack.pop()

    a = exp_stack.pop()

    b = exp_stack.pop()

    exp_stack.append(op + b + a)

print(fPREFIX: {exp_stack[-1]}'')

return exp_stack[-1]

expres = input("INPUT THE EXPRESSION: ")

pre = infix_to_prefix(expres)

pos = infix_to_postfix(expres)

```

Output:

```

INPUT THE EXPRESSION: A+B-C*D
PREFIX: -+AB*CD
POSTFIX: AB+CD*-
...Program finished with exit code 0
Press ENTER to exit console. []

```

Result:

Thus the program was successfully compiled and run.

S.AKANSHA

RA1911029010060 CSE CN P1

Exp 11: Intermediate code generation – Quadruple, Triple, Indirect triple

Aim:

A program to implement intermediate code generation - Quadruple, Triple, Indirect triple.

Algorithm:

The algorithm takes a sequence of three-address statements as input. For each three address statements of the form $a := b \text{ op } c$ perform the various actions. These are as follows:

1. Invoke a function getreg to find out the location L where the result of computation $b \text{ op } c$ should be stored.
2. Consult the address description for y to determine y' . If the value of y currently in memory and register both then prefer the register y' . If the value of y is not already in L then generate the instruction $\text{MOV } y', L$ to place a copy of y in L.
3. Generate the instruction $\text{OP } z', L$ 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 $x := y \text{ op } z$ those register will no longer contain y or z.

Program:

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

```

void small();
void dove(int i);
int p[5]={0,1,2,3,4},c=1,i,k,l,m,pi;
char sw[5]={'=','-','+','/','*'},{j[20],a[5],b[5],ch[2];
void main()
{
    printf("Enter the expression : ");
    scanf("%s",j);
    printf("The Intermediate code is :\n");
    small();
}

void dove(int i)
{
    a[0]=b[0]='\0';
    if(!isdigit(j[i+2])&&!isdigit(j[i-2]))
    {
        a[0]=j[i-1];
        b[0]=j[i+1];
    }
    if(isdigit(j[i+2]))
    {
        a[0]=j[i-1];
        b[0]='t';
        b[1]=j[i+2];
    }
    if(isdigit(j[i-2]))
    {
        b[0]=j[i+1];
        a[0]='t';
        a[1]=j[i-2];
        b[1]='\0';
    }
}

```

```

}

if(isdigit(j[i+2]) &&isdigit(j[i-2]))
{
    a[0]='t';
    b[0]='t';
    a[1]=j[i-2];
    b[1]=j[i+2];
    sprintf(ch,"%d",c);
    j[i+2]=j[i-2]=ch[0];
}

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

void small()
{
    pi=0;l=0;
    for(i=0;i<strlen(j);i++)
    {
        for(m=0;m<5;m++)
        if(j[i]==sw[m])

```

```
if(pi<=p[m])
{
    pi=p[m];
    l=1;
    k=i;
}
if(l==1)
dove(k);
else
exit(0);
}
```

Input:

a=b+c-d

Output:

```
Enter the expression : a=b+c-d
The Intermediate code is :
t1=b+c
t2=t1-d
a=t2
-----
Process exited after 4.008 seconds with return value 0
Press any key to continue . . .
```

Result:

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