

**SRM INSTITUTE OF SCIENCE & TECHNOLOGY, NCR
CAMPUS**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB RECORD FILE

Name : Palak Khurana
Registration No : RA1911003030437
Section : I
Subject Code : 18CSC304J
Title : COMPILER DESIGN LAB
Semester : VI

Academic year 2021 - 2022



SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

DELHI-NCR CAMPUS, GHAZIABAD (U.P)

**SRM INSTITUTE OF SCIENCE & TECHNOLOGY, NCR CAMPUS,
MODINAGAR, GHAZIABAD, U.P – 201204**

Register No.:

RA1911003030437

BONAFIDE CERTIFICATE

This is to certify that Lab Report of **Compiler Design Laboratory (18CSC304J)**, which is submitted by **Palak Khurana** in partial fulfillment of the requirement for the award of degree of B. Tech. in Department of Computer Science & Engineering of SRM Institute of Science & Technology, NCR Campus, Modinagar, is a record of the candidate own work carried out by him/her under our supervision during the academic year 2019-2020.

 **SRM**
INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)
DELHI-NCR CAMPUS, GHAZIABAD (U.P)

HEAD OF DEPARTMENT

LAB IN-CHARGE

Submitted for the university examination held on_____.

INDEX

Exp. No.	Experiment Name	Date of Conduction	Date of Submission	Faculty Signature
1.	Transition Diagram to Transition table			
2.	Implementation of lexical Analyzer.			
3.	Computation of FIRST sets			
4.	Computation of FOLLOW sets			
5.	Intermediate Code Generation			
6.	Implementation of Shift Reduce Parsing			
7.	Computation of LEADING sets			
8.	Computation of TRAILING sets			
9.	Intermediate Code Generation: Postfix Expression			
10.	Intermediate Code Generation: Prefix Expression			
11.	Construction of Directed Acyclic Graph			
12.	Implementation of Recursive Descent Parsing			

Experiment 1 : Transition Diagram to Transition Table

Aim: Write a program in C/C++ to show the transition table from a given transition diagram.

Algorithm:

1. Start
2. Enter the number of states.
3. Enter the number of input variables.
4. Enter the state and its information.
5. Enter the input variables.
6. Enter the transition function information i.e., transition value from a state with an input variable.
7. Show the Transition Table.
8. Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct setStates
{
    int state;
    int final; // 0 - NO    1 - YES
    int start; // 0 - NO    1 - YES
};
typedef struct setStates sstate;

void main()
{
    int s,v,i,j;
    int **sv,*var;
    sstate *states;

    printf("\nInput the number of finite set of states : ");
    scanf("%d",&s);
    printf("\nInput the number of finite set of input variables : ");
    scanf("%d",&v);

    // creating transition table
    sv = (int **)malloc(v*sizeof(int));
    //printf("\n1 sucess\n");
    for(i=0;i<s;i++)
    {
        sv[i]=(int *)malloc(sizeof(int));
```

```

    }
    /*printf("\n2 sucess\n");
    printf("\nThe Array : \n");
    for(i=0;i<s;i++)
    {
        for(j=0;j<v;j++)
        {
            printf("%d\t",sv[i][j]);
        }
        printf("\n");
    }*/

    // storing state information
    states = (sstate *)malloc(s*sizeof(sstate));
    printf("\nInput the states and its info (state start final): \n");
    for(i=0;i<s;i++)
    {
        scanf("%d%d%d",&states[i].state,&states[i].start,&states[i].final);
    }

    // storing input veribale
    var = (int *)malloc(v*sizeof(int));
    printf("\nInput the variables : \n");
    for(i=0;i<v;i++)
    {
        scanf("%d",&var[i]);
    }

    // storing inputs of transition function
    for(i=0;i<s;i++)
    {
        for(j=0;j<v;j++)
        {
            printf("\nThe sates %c with input veribale %c move to state : 
",states[i].state,var[j]);
            scanf("%d",&sv[i][j]);
        }
    }

    // display transition table on screen
    printf("\nThe Transition Table : \n");
    printf("\t");
    for(i=0;i<v;i++)
    {
        printf("%c\t",var[i]);
    }
    printf("\n.....");
    for(i=0;i<s;i++)
    {

```

```

printf("\n%c %c %c\t",states[i].state,(states[i].start==0)? ' ': '$',(states[i].final==0)?
': '*');
for(j=0;j<v;j++)
{
    printf("%c\t",sv[i][j]);
}
printf("\n");
}
}

```

Output:

```
PS C:\CD Lab> cd "c:\CD Lab\" ; if ($?) { gcc Lab1.c -o Lab1 } ; if ($?) { .\Lab1 }
```

```
Input the number of finite set of states : 4
```

```
Input the number of finite set of input variables : 2
```

```
Input the states and its info (state start final):
```

```
98 1 1
97 0 0
99 0 0
100 0 0
```

```
Input the variables :
```

```
48
49
```

```
The states b with input variable 0 move to state : 100
```

```
The states b with input variable 1 move to state : 97
```

```
The states a with input variable 0 move to state : 98
```

```
The states a with input variable 1 move to state : 100
```

```
The states c with input variable 0 move to state : 97
```

```
The states c with input variable 1 move to state : 100
```

```
The states d with input variable 0 move to state : 87
```

```
The states d with input variable 1 move to state : 100
```

```
The Transition Table :
```

```

      0      1
-----
b $ * d      a
a      b      d
c      a      d
d      W      d
PS C:\CD Lab>

```

Result : The Program executed successfully .

Experiment 2: Implementation of Lexical Analyzer

Aim: Write a program in C/C++ to implement a lexical analyzer.

Algorithm:

1. Start
2. Get the input expression from the user.
3. Store the keywords and operators.
4. Perform analysis of the tokens based on the ASCII values.
- 5.

<u>ASCII Range</u>	<u>TOKEN TYPE</u>
97-122	Keyword else identifier
48-57	Constant else operator
Greater than 12	Symbol

6. Print the token types.
7. Stop

Program:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
void main()
{
    char
key[11][10]={ "for","while","do","then","else","break","switch","case","if","continue"};
    char oper[13]={ '+','-','*','/','%','&','<','>','=',';',':', '!'};
    char a[20],b[20],c[20];
    int i,j,l,m,k,flag;
    clrscr();
    printf("\n Enter the expression: ");
    gets(a);
    i=0;
    while(a[i])
    {
        flag=0;
        j=0;
        l=0;
        b[0]='\0';
        if((toascii(a[i])>=97))&&(toascii(a[i])<=122)))
        {
            if((toascii(a[i+1])>=97))&&(toascii(a[i+1])<=122)))
            {
                while((toascii(a[i])>=97))&&(toascii(a[i])<=122)))
```

```

        {
            b[j]=a[i];
            j++; i++;
        }
        b[j]='\0';
    }
    else
    {
        b[j]=a[i];
        i++;
        b[j+1]='\0';
    }
    for(k=0;k<=9;k++)
    {
        if(strcmpi(b,key[k])==0)
        {
            flag=1;
            break;
        }
    }

    if(flag==1)
        printf("\n %s is the keyword",b);
    else
        printf("\n %s is the identifier",b);
}
else if((toascii(a[i])>=48)&&(toascii(a[i])<=57)))
{
    if((toascii(a[i+1])>=48)&&(toascii(a[i+1])<=57)))
    {
        while((toascii(a[i])>=48)&&(toascii(a[i])<=57)))
        {
            c[l]=a[i];
            l++; i++;
        }
    }
    else
    {
        c[l]=a[i];
        i++; l++;
    }
    c[l]='\0';
    printf("\n %s is the constant",c);
}
//second ifelse
else
{
    for(m=0;m<13;m++)
    {
        if(a[i]==oper[m])

```



```

        {
            printf("\n %c is the operator",a[i]);
            break;
        }
    }
    if(m>=13)
        printf("\n %c is the symbol",a[i]);
    i++;
} //last else
} //while
getch();
}

```

Output:

```

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

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

PS C:\Compiler Design> cd "C:\Compiler Design\" ; if ($?) { gcc lexicalAnalyser.c -o lexicalAnalyser } ; if ($?) { .\lexicalAnalyser }
Enter the expression: if(a<20)break

if is the keyword
( is the symbol
a is the identifier
< is the operator
20 is the constant
) is the symbol
break is the keyword
PS C:\Compiler Design>

```

Result : The Program executed successfully .

Experiment 3: Computation of FIRST Sets

Aim: Write a program in C/C++ to find the FIRST set for a given set of production rule of a grammar.

Algorithm:

Procedure First

1. Input the number of production N.
2. Input all the production rule *PArray*
3. Repeat steps a, b, c until process all input production rule i.e. *PArray*[N]
 - a. If $X_i \neq X_{i+1}$ then
 - i. Print Result array of X_i which contain FIRST(X_i)
 - b. If first element of X_i of *PArray* is Terminal or ϵ Then
 - i. Add Result = Result U first element
 - c. If first element of X_i of *PArray* is Non-Terminal Then
 - i. searchFirst(i, *PArray*, N)
4. End Loop
5. If N (last production) then
 - a. Print Result array of X_i which contain FIRST(X_i)
6. End

Procedure searchFirst(i, *PArray*, N)

1. Repeat steps Loop $j=i+1$ to N
 - a. If first element of X_j of *PArray* is Non-Terminal Then
 - i. searchFirst(j, of *PArray*, N)
 - b. If first element of X_j of *PArray* is Terminal or ϵ Then
 - i. Add Result = Result U first element
 - ii. Flag=0
2. End Loop
3. If Flag = 0 Then
 - a. Print Result array of X_j which contain FIRST(X_j)
4. End

Program:

```
#include<bits/stdc++.h>
using namespace std;
```

```
void searchFirst(int n, int i, char pl[], char r[], char result[], int k)
{
    int j,flag;
    for(j=i+1;j<n;j++)
    {
```

```

if(r[i]==pl[j])
{
if(isupper(r[j]))
{
searchFirst(n,j,pl,r,result,k);
}
if(islower(r[j]) || r[j]=='+' || r[j]=='*' || r[j]=='(' || r[j]==')')
{
result[k++]=r[j]; result[k++]='';
flag=0;
}
}
}
if(flag==0)
{
for(j=0;j<k-1;j++){
cout<<result[j];
}
}
}
int main()
{
char pr[10][10],pl[10],r[10],prev,result[10];
int i,n,k,j;
cout<<"How many production rule : ";
cin>>n;
if(n==0)
exit(0);
for(i=0;i<n;i++)
{
cout<<"\nInput left part of production rules : ";
cin>>pl[i];
cout<<"\nInput right part of production rules : ";
cin>>pr[i];
r[i]=pr[i][0];
}
cout<<"\nProduction Rules are : \n";
for(i=0;i<n;i++)
{
cout<<pl[i]<<"->"<<pr[i]<<"\n";
}
cout<<"\n----O U T P U T---\n\n";
prev=pl[0];
k=0;
for(i=0;i<n;i++)
{
if(prev!=pl[i])
{
cout<<"\nFIRST("<<prev<<")={ ";

```

```

for(j=0;j<k-1;j++)
cout<<result[j];
cout<<" ";
k=0;
prev=pl[i];
}
if(prev==pl[i])
{
if(islower(r[i]) || r[i]== '+' || r[i]=='*' || r[i]=='') || r[i]=='(')
{
result[k++]=r[i]; result[k++]='';
}
if(isupper(r[i]))
{
cout<<"\nFIRST("<<prev<<")={ ";
searchFirst(n,i,pl,r,result,k);
cout<<" ";
k=0;
prev=pl[i+1];
}
}
}
if(i==n){
cout<<"\nFIRST("<<prev<<")={ ";
for(j=0;j<k-1;j++)
cout<<result[j];
cout<<" ";
k=0;
prev=pl[i];
}
return 0;
}

```

Output:

```
PS C:\CD Lab> cd "c:\CD Lab\" ; if ($?) { g++ First.cpp -o First } ; if ($?) { .\First }
How many production rule : 5

Input left part of production rules : E
Input right part of production rules : aTX
Input left part of production rules : E
Input right part of production rules : TX
Input left part of production rules : T
Input right part of production rules : FY
Input left part of production rules : F
Input right part of production rules : (E)
Input left part of production rules : F
Input right part of production rules : i

Production Rules are :
E->aTX
E->TX
T->>FY
F->(E)
F->i

----O U T P U T---

FIRST(E)={a,(,i}
FIRST(T)={(,i}
FIRST(F)={(,i}
PS C:\CD Lab> █
```

Result : The Program executed successfully .

Experiment 4: Computation of FOLLOW Sets

Aim: Write a program in C/C++ to find a FOLLOW set from a given set of production rule.

Algorithm:

1. Declare the variables.
 2. Enter the production rules for the grammar.
 3. Calculate the FOLLOW set for each element call the user defined function follow().
 4. If $x \rightarrow aBb$
 - a. If x is start symbol then $FOLLOW(x) = \{ \$ \}$.
 - b. If b is NULL then $FOLLOW(B) = FOLLOW(x)$.
 - c. If b is not NULL then $FOLLOW(B) = FIRST(b)$.
- END.

Program:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
int n,m=0,p,i=0,j=0;
char a[10][10],f[10];
void follow(char c);
void first(char c);
int main()
{
int i,z;
char c,ch;
printf("Enter the no.of productions:");
scanf("%d",&n);
printf("Enter the productions(epsilon=$):\n");
for(i=0;i<n;i++)
scanf("%s%c",a[i],&ch);
do
{
m=0;
printf("Enter the element whose FOLLOW is to be found:");
scanf("%c",&c);
follow(c);
printf("FOLLOW(%c) = { ",c);
for(i=0; i<m; i++)
printf("%c ",f[i]);
printf(" }\n");
printf("Do you want to continue(0/1)?");
scanf("%d%c",&z,&ch);
}
while(z==1);
return 0;
```

```

}
void follow(char c)
{

if(a[0][0]==c)
f[m++]='$';
for(i=0;i<n;i++)
{
for(j=2;j<strlen(a[i]);j++)
{
if(a[i][j]==c)
{
if(a[i][j+1]!='\0')
first(a[i][j+1]);
if(a[i][j+1]=='\0'&&c!=a[i][0])
follow(a[i][0]);
}
}
}
}
void first(char c)
{
int k;
if(!(isupper(c)))
f[m++] = c;
for(k=0;k<n;k++)
{
if(a[k][0]==c)
{
if(a[k][2]=='$')
follow(a[i][0]);
else if(islower(a[k][2]))
f[m++] = a[k][2];
else
first(a[k][2]);
}
}
}
}

```

Output:

```
PS C:\CD Lab> cd "c:\CD Lab\" ; if ($?) { g++ Follow.cpp -o Follow } ; if ($?) { .\Follow }
Enter the no.of productions:5
Enter the productions(epsilon=$):
S=(L)
S=a
L=SF
F=,SF
F=$
Enter the element whose FOLLOW is to be found:S
FOLLOW(S) = { $ , ) }
Do you want to continue(0/1)?1
Enter the element whose FOLLOW is to be found:L
FOLLOW(L) = { ) }
Do you want to continue(0/1)?F
Enter the element whose FOLLOW is to be found:FOLLOW(F) = { ) }
Do you want to continue(0/1)?0
PS C:\CD Lab> █
```

Result : The Program executed successfully .

Experiment 5: Intermediate Code Generation

Aim: Write a program in C/C++ to generate intermediate code from a given syntax tree statement.

Algorithm:

1. Start the process.
 2. Input an expression EXP from user.
 3. Process the expression from right hand side to left hand side.
 4. FLAG:=0; TOP = -1;
 5. IF EXP = '=' then
 - i. IF EXP(index - 1) = 0 then
 1. PRINT EXP element from index to (index - 1) and POP STACK[TOP]. Terminate
 - Else
 - i. PRINT Wrong Expression
- [EndIF]
- IF an operator is found and FLAG = 0 then
- i. TOP:= TOP + 1
 - ii. add to STACK[TOP].
 - iii. FLAG:=1
- Else
- i. pop twice the STACK and result add to the newID(identifier) and PRINT.
 - ii. TOP:=TOP-2. Save newID to STACK[TOP]
 - iii. FLAG:=0
- [EndIF]

6. IF an operand is found then
 - i. TOP:=TOP+1
 - ii. move to STACK [TOP]
 - iii. IF TOP > 1 then
 1. pop twice the STACK and result add to the newID(identifier) and PRINT.
 2. TOP:=TOP-2. Save newID to STACK[TOP]
 3. FLAG:=0
- [End]
7. End the process

Program:

```
#include<iostream>
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
using namespace std;
int main()
{
char g,exp[20],stack[20];
int m=0,i,top=-1,flag=0,len,j;
cout<<"\nInput an expression : ";
gets(exp);
cout<<"\nIntermediate code generator\n";
len=strlen(exp);
if(isdigit(exp[len-1]))
{
cout<<"T = inttoreal(";
i=len-1;
while(isdigit(exp[i]))
{
i--;
}
for(j=i+1;j<len;j++)
{
cout<<exp[j];
}
cout<<".0)\n";
exp[i+1]='T';
len=i+2;
}
else
{
cout<<"T = "<<exp[len-1]<<"\n";
exp[len-1]='T';
```

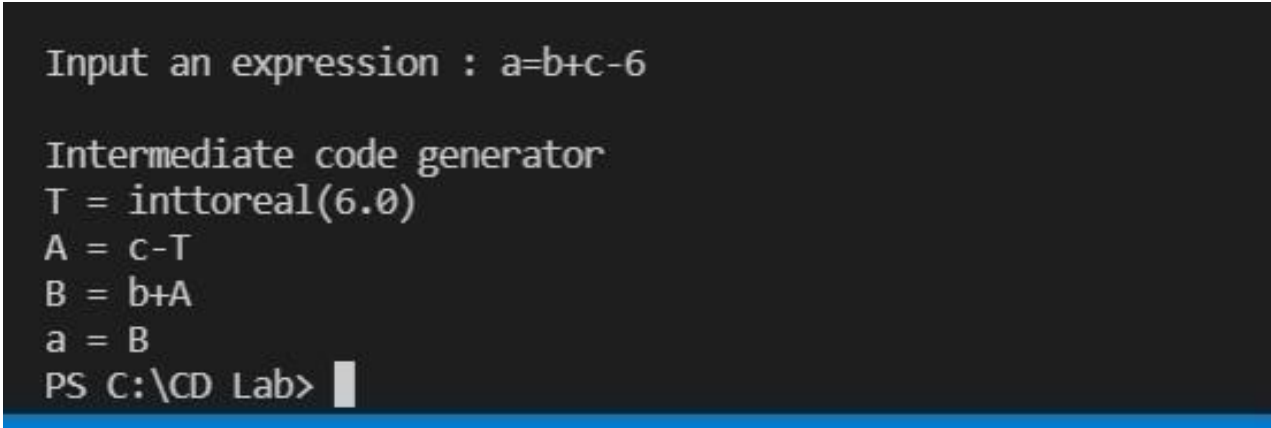
```

}
for(i=len-1;i>=0;i--)
{
if(exp[i]=='=')
{
if((i-1)==0)
{
if(isalpha(stack[top]))
{
cout<<exp[i-1]<<" "<<exp[i]<<" "<<stack[top];
}
else
{
cout<<exp[i-1]<<" "<<exp[i]<<" "<<stack[top]<<stack[top-1];
}
break;
}
else
{
cout<<"\nWrong Expression !";
break;
}
}
if(exp[i]=='+'||exp[i]=='/'||exp[i]=='*'||exp[i]=='-'||exp[i]=='%')
{
if(flag==0)
{
flag=1;
top=top+1;
stack[top]=exp[i];
}
else
{
g=char('A' + m);
m++;
cout<<g<<" = "<<stack[top]<<stack[top-1]<<"\n";
stack[top-1]=g;
stack[top]=exp[i];
flag=0;
}
}
else
{
top=top+1;
stack[top]=exp[i];
if(top>1)
{
g=char('A' + m);m++;

```

```
cout<<g<<" = "<<stack[top]<<stack[top-1]<<stack[top-2]<<"\n";
top=top-2;
stack[top]=g;
flag=0;
}
}
}
return 0;
```

Output:



```
Input an expression : a=b+c-6
```

```
Intermediate code generator
```

```
T = inttoreal(6.0)
```

```
A = c-T
```

```
B = b+A
```

```
a = B
```

```
PS C:\CD Lab> █
```

Result : The Program executed successfully .

Experiment 6: Implementation of Shift Reduce Parsing

Aim: Write a program in C/C++ to implement the shift reduce parsing.

Algorithm:

1. Start the Process.
2. Symbols from the input are shifted onto stack until a handle appears on top of the stack.
3. The Symbols that are the handle on top of the stack are then replaces by the left-hand side of the production (reduced).
4. If this result in another handle on top of the stack, then another reduction is done, otherwise we go back to shifting.
5. This combination of shifting input symbols onto the stack and reducing productions when handles appear on the top of the stack continues until all of the input is consumed and the goal symbol is the only thing on the stack - the input is then accepted.
6. If we reach the end of the input and cannot reduce the stack to the goal symbol, the input is rejected.
7. Stop the process.

Program :

```
#include <bits/stdc++.h>
using namespace std;
int z = 0, i = 0, j = 0, c = 0;
char a[16], ac[20], stk[15], act[10];
void check()
{
    strcpy(ac, "REDUCE TO E -> ");
    for(z = 0; z < c; z++)
    {
        if(stk[z] == '4')
        {
            printf("%s4", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            printf("\n%s\t%s\t", stk, a);
        }
    }
    for(z = 0; z < c - 2; z++)
    {
        if(stk[z] == '2' && stk[z + 1] == 'E' &&
            stk[z + 2] == '2')
        {
            printf("%s2E2", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
```

```

        stk[z + 2] = '\0';
        printf("\n$$s\t%s$\t", stk, a);
        i = i - 2;
    }
}
for(z = 0; z < c - 2; z++)
{
    if(stk[z] == '3' && stk[z + 1] == 'E' &&
    stk[z + 2] == '3')
    {
        printf("%s3E3", ac);
        stk[z]='E';
        stk[z + 1]='\0';
        stk[z + 1]='\0';
        printf("\n$$s\t%s$\t", stk, a);
        i = i - 2;
    }
}
return ;
}
int main()
{
    printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");
    strcpy(a,"32423");
    c=strlen(a);
    strcpy(act,"SHIFT");
    printf("\nstack \t input \t action");
    printf("\n$$\t%s$\t", a);
    for(i = 0; j < c; i++, j++)
    {
        printf("%s", act);
        stk[i] = a[j];
        stk[i + 1] = '\0';
        a[j]=' ';
    }
    printf("\n$$s\t%s$\t", stk, a);
    check();
}
check();
if(stk[0] == 'E' && stk[1] == '\0')
    printf("Accept\n");
else
    printf("Reject\n");
}

```

Output:

GRAMMAR is -

$E \rightarrow 2E2$

$E \rightarrow 3E3$

$E \rightarrow 4$

stack	input	action
\$	32423\$	SHIFT
\$3	2423\$	SHIFT
\$32	423\$	SHIFT
\$324	23\$	REDUCE TO $E \rightarrow 4$
\$32E	23\$	SHIFT
\$32E2	3\$	REDUCE TO $E \rightarrow 2E2$
\$3E	3\$	SHIFT
\$3E3	\$	REDUCE TO $E \rightarrow 3E3$
\$E	\$	Accept

PS C:\CD Lab>

Result : The Program executed successfully .

Experiment 6: Computation of Leading Sets

Aim: Write a program in C/C++ to detect the leading edges of the given set of productions of a grammar.

Algorithm:

1. Start the program.
2. Get the Set of Productions for the grammar from the user. No redundant & cyclic productions must be given.
3. The conditions to be checked are:

Conditions

>Sa
S->Aa
S->ab
S->AB
S->SA
S->a
S->SA*
S->*a

Inclusions in resultS-

add a
add a, production of A
add a
Production of A
none
take a
none taken
take * leave a

4. Print the Leading edges.
5. Stop the program.

Program :

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

int vars, terms,i,j,k,m,rep,count,temp=-1;
char var[10],term[10],lead[10][10];
struct grammar
{
    int prodno;
    char lhs,rhs[20][20];
}gram[50];
void get()
{
    cout<<"\nLEADING\n";
    cout<<"\nEnter the no. of variables : ";
    cin>>vars;
    cout<<"\nEnter the variables : \n";
```



```

for(i=0;i<vars;i++)
{
    cin>>gram[i].lhs;
    var[i]=gram[i].lhs;
}
cout<<"\nEnter the no. of terminals : ";
cin>>terms;
cout<<"\nEnter the terminals : ";
for(j=0;j<terms;j++)
    cin>>term[j];
cout<<"\nPRODUCTION DETAILS\n";
for(i=0;i<vars;i++)
{
    cout<<"\nEnter the no. of production of "<<gram[i].lhs<<":";
    cin>>gram[i].prodno;
    for(j=0;j<gram[i].prodno;j++)
    {
        cout<<gram[i].lhs<<"->";
        cin>>gram[i].rhs[j];
    }
}
}
void leading()
{
    for(i=0;i<vars;i++)
    {
        for(j=0;j<gram[i].prodno;j++)
        {
            for(k=0;k<terms;k++)
            {
                if(gram[i].rhs[j][0]==term[k])
                    lead[i][k]=1;
                else
                {
                    if(gram[i].rhs[j][1]==term[k])
                        lead[i][k]=1;
                }
            }
        }
    }
    for(rep=0;rep<vars;rep++)
    {
        for(i=0;i<vars;i++)
        {
            for(j=0;j<gram[i].prodno;j++)
            {
                for(m=1;m<vars;m++)
                {
                    if(gram[i].rhs[j][0]==var[m])

```

```

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

void 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;
    }
}

int main()
{
    get();
    leading();
    display();
}

```

Output:

```
PS C:\CD Lab> cd "c:\CD Lab\" ; if ($?) { g++ Leading.cpp -o Leading } ; if ($?) { .\Leading }

LEADING

Enter the no. of variables : 3

Enter the variables :
E T F

Enter the no. of terminals : 5

Enter the terminals : ( ) * + i

PRODUCTION DETAILS

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

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

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

LEADING(E) = (,*,+,i,
LEADING(T) = (,*,i,
LEADING(F) = (,i,
PS C:\CD Lab> █
```

Result : The Program executed successfully .

Experiment 8: Computation of Trailing Sets

Aim: Write a program in C/C++ or Java to detect the trailing edges of the given set of productions of a grammar.

Algorithm:

1. Start the program.
2. Get the Set of Productions for the grammar from the user. No redundant & cyclic productions must be given.
3. Reverse each input productions and print it.
4. The conditions to be checked according to the reversed inputs

<u>are:Conditions</u>	<u>Inclusions in result</u>
S->Sa	add a
S->Aa	add a, production of A
S->ab	add a
S->AB	Production of A
S->SA	none
S->a	take a
S->SA*	none taken
S->*a	take * leave a

5. Print the Trailing edges.
6. Stop the program.

Program :

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

int vars, terms ,i ,j, k, m, rep, count, temp=-1;
char var[10],term [10],lead[10][10],trail[10][10];
struct grammar
{
    int prodno;
    char lhs,rhs[20][20];
}gram[50];
void get()
{
    cout<<"\nTRAILING\n";
    cout<<"\nEnter the no. of variables : ";
    cin>>vars;
    cout<<"\nEnter the variables : \n";
    for(i=0;i<vars;i++)
```

```

{
    cin>>gram[i].lhs;
    var[i]=gram[i].lhs;
}
cout<<"\nEnter the no. of terminals : ";
cin>>terms;
cout<<"\nEnter the terminals : ";
for(j=0;j<terms;j++)
    cin>>term[j];
cout<<"\nPRODUCTION DETAILS\n";
for(i=0;i<vars;i++)
{
    cout<<"\nEnter the no. of production of "<<gram[i].lhs<<": ";
    cin>>gram[i].prodno;
    for(j=0;j<gram[i].prodno;j++)
    {
        cout<<gram[i].lhs<<"->";
        cin>>gram[i].rhs[j];
    }
}
}
void 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<<"\nTRAILING("<<gram[i].lhs<<") = ";
        for(j=0;j<terms;j++)
        {
            if(trail[i][j]==1)
                cout<<term[j]<<",";

        }
        cout<<endl;
    }
}

int main()
{
    get();
    trailing();
    display();

}

```

Output :

TRAILING

Enter the no. of variables : 3

Enter the variables :

E T F

Enter the no. of terminals : 5

Enter the terminals : () * + i

PRODUCTION DETAILS

Enter the no. of production of E:2

E->E+T

E->T

Enter the no. of production of T:2

T->T*F

T->F

Enter the no. of production of F:2

F->(E)

F->i

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

TRAILING(T) =),*,i,

TRAILING(F) =),i,

PS C:\CD Lab> █

Result : The Program executed successfully .

Experiment 9 : Intermediate code generation - Postfix expression

Aim: Write a program in C/C++ or Java to generate Intermediate Code (Postfix Expression) from given syntax tree.

Algorithm:

1. Declare a 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 discards both parentheses.
10. Pop and output from the stack until it is not empty.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define SIZE 100
char stack[SIZE];
int top = -1;
void push(char item)
{
    if(top >= SIZE-1)
    {
        printf("\nStack Overflow.");
    }
    else
    {
        top = top+1;
        stack[top] = item;
```



```

}
}
char pop()
{
char item ;
if(top <0)
{
printf("stack under flow: invalid infix expression");
getchar();
exit(1);
}
else
{
item = stack[top];
top = top-1;
return(item);
}
}
int is_operator(char symbol)
{
if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
{
return 1;
}
else
{
return 0;
}
}
int precedence(char symbol)
{
if(symbol == '^')
{
return(3);
}
else if(symbol == '*' || symbol == '/')
{
return(2);
}
else if(symbol == '+' || symbol == '-')
{
return(1);
}
else
{
return(0);
}
}
void InfixToPostfix(char infix_exp[], char postfix_exp[])

```

```

{
int i, j;
char item;
char x;
push('(');
strcat(infix_exp, "");
i=0;
j=0;
item=infix_exp[i];
while(item != '\0')
{
if(item == '(')
{
push(item);
}
else if( isdigit(item) || isalpha(item))
{
postfix_exp[j] = item;
j++;
}
else if(is_operator(item) == 1)
{
x=pop();
while(is_operator(x) == 1 && precedence(x)>= precedence(item))
{
postfix_exp[j] = x;
j++;
x = pop();
}
push(x);
push(item);
}
else if(item == ')')
{
x = pop();
while(x != '(')
{
postfix_exp[j] = x;
j++;
x = pop();
}
}
else
{
printf("\nInvalid infix Expression.\n");
getchar();
exit(1);
}
i++;
}

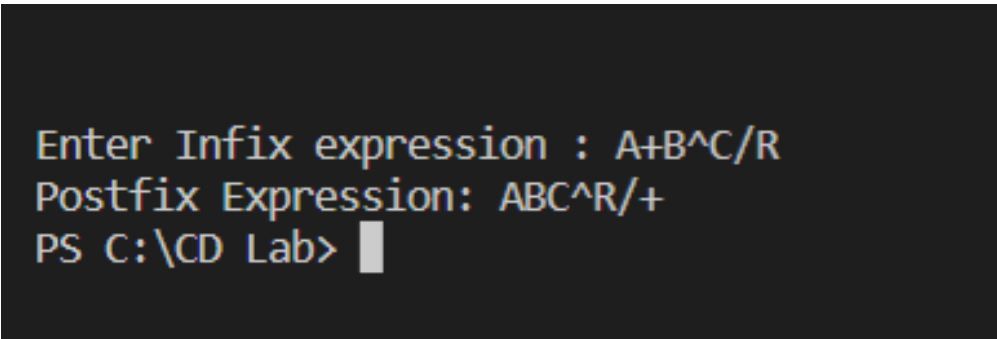
```

```

item = infix_exp[i];
}
if(top>0)
{
printf("\nInvalid infix Expression.\n");
getchar();
exit(1);
}
if(top>0)
{
printf("\nInvalid infix Expression.\n");
getchar();
exit(1);
}
postfix_exp[j] = '\0';
}
int main()
{
char infix[SIZE], postfix[SIZE];
printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");
printf("\nEnter Infix expression : ");
gets(infix);
InfixToPostfix(infix,postfix);
printf("Postfix Expression: ");
puts(postfix);
return 0;
}

```

Output :



```

Enter Infix expression : A+B^C/R
Postfix Expression: ABC^R/+
PS C:\CD Lab>

```

Result : The Program executed successfully .

Experiment 10: Intermediate code generation - Prefix Expression

Aim: Write a program in C/C++ or Java to generate Intermediate Code (Prefix Expression) from given syntax tree.

Algorithm :

1. First, reverse the given infix expression.
2. Scan the characters one by one.
3. If the character is an operand, copy it to the prefix notation output.
4. If the character is a closing parenthesis, then push it to the stack.
5. If the character is an opening parenthesis, pop the elements in the stack until we find the corresponding closing parenthesis.
6. If the character scanned is an operator
 - If the operator has precedence greater than or equal to the top of the stack, push the operator to the stack.
 - If the operator has precedence lesser than the top of the stack, pop the operator and output it to the prefix notation output and then check the above condition again with the new top of the stack.
7. After all the characters are scanned, reverse the prefix notation output.

Program:

```
#define SIZE 50 /* Size of Stack */  
  
#include<string.h>  
  
#include<stdio.h>  
  
#include <ctype.h>  
  
char s[SIZE];  
  
int top=-1;  
  
char push(char elem)  
  
{ /* Function for PUSH operation */
```

```

s[++top]=elem;

}

char pop()

{ /* Function for POP operation */

return(s[top--]);

}

int pr(char elem)

{ /* Function for precedence */

switch(elem)

{

case '#': return 0;

case ')': return 1;

case '+':

case '-': return 2;

case '*':

case '/': return 3;

}

}

int main()

{ /* Main Program */

char infx[50],prfx[50],ch,elem;

int i=0,k=0;

printf("\nRead the Infix Expression : ");

scanf("%s",infx);

push('#');

strev(infx);

while( (ch=infx[i++]) != '\0')

```

```

{
if( ch == ')')
push(ch);
else
if(isalnum(ch))
prfx[k++]=ch;
else
if( ch == '(')
{
while( s[top] != ')')
prfx[k++]=pop();
elem=pop(); /* Remove ) */
}
else
{ /* Operator */
while( pr(s[top]) >= pr(ch) )
prfx[k++]=pop();
push(ch);
}
}

while( s[top] != '#') /* Pop from stack till empty */
prfx[k++]=pop();
prfx[k]='\0'; /* Make prfx as valid string */
strrev(prfx);
strrev(infx);
printf("\n\nGiven Infix Expression: %s\n Prefix Expn: %s\n",infx,prfx);
return 0;

```

}

Output:

```
Read the Infix Expression : (a+b)*c
```

```
Given Infix Expression: (a+b)*c
```

```
Prefix Expn: *+abc
```

```
PS C:\CD Lab> █
```

Result : The Program executed successfully .

Experiment 11: Construction of DAG

Aim: Write a c or c++ or java to Construct DAG for input expression.

Program:

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string exp;
    cout<<"Enter the expression: ";
    cin>>exp;
    int j=0,k=0;
    char q;
    for(int i=exp.length()-1;i>1;i--)
    {
        if(islower(exp[i]) || (exp[i]>=48 && exp[i]<=57))
        {
            cout<<j<<"-"<<exp[i]<<endl;
            j++;
        }
    }
    for(int i=exp.length()-1;i>1;i--)
    {
        if(!(islower(exp[i]) || (exp[i]>=48 && exp[i]<=57)))
        {
            cout<<j<<"-"<<exp[i]<<k<<k+1<<endl;
            j++;
            k+=2;
        }
    }
    cout<<j<<"-"<<exp[0]<<endl;
    j++;
    cout<<j<<"-"<<exp[1]<<j-1<<j-2<<endl;
    return 0;
}
```


Output:

```
PS C:\CD Lab> cd "c:\CD Lab\" ; if ($?) { g++ DAG.cpp -o DAG } ; if ($?) { .\DAG }  
Enter the expression: a=b+c-5  
0->5  
1->c  
2->b  
3->-01  
4->+23  
5->a  
6->=54
```

Result: The program executed successfully.

Experiment 12: Recursive Descent Parsing

Aim: Write a program in C/ C++ or Java to implement Recursive Descent Parsing.

Program:

```
#include<iostream>
#include<map>
#include<vector>
using namespace std;
int main()
{
    int flag = 0;
    map<char,vector<string> >rules;
    string exp,test;
    rules['S'].push_back("aAc");
    rules['A'].push_back("cd");
    rules['A'].push_back("d");
    cout<<"Enter the string: ";
    cin>>exp;
    string start="aAc";
    if(start[0]!=exp[0])
    cout<<"Not Accepted";
    else
    {
        cout<<"S"<<endl<<start<<endl;
        string a= (rules['A'])[0];
        string b=(rules['A'])[1];
        string t;
        t=start[0]+a+start[2];
        cout<<t<<endl;
        if(t==exp)
        {
            flag = 1;
            cout<<"Accepted";
        }
        else
        {
            cout<<start<<endl;
            t=start[0]+b+start[2];
            cout<<t<<endl;
            if(t==exp)
            {
                flag = 1;
                cout<<"Accepted";
            }
        }
    }
}
```

```
f(flag == 0)
cout<<"Not accepted";
return 0;
}
```

Output:

```
PS C:\CD Lab> cd "c:\CD Lab\" ; if ($?) { g++ recursiveDescent.cpp -o recursiveDescent } ; if ($?) { .\recursiveDescent }
Enter the string: adc
S
aAc
acdc
aAc
adc
Accepted
PS C:\CD Lab> █
```

Result: The program executed successfully.