# COMPILER DESIGN (18CSC304J)

## B.Tech (CSE) – 3rd year/6th Semester

Name:

Registration no:

## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

## FACULTY OF ENGINEERING & TECHNOLOGY


SRM INSTITUTE OF SCIENCE & TECHNOLOGY,

Delhi NCR CAMPUS, MODINAGAR

SIKRI KALAN, DELHI MEERUT ROAD, DIST. – GHAZIABAD - 201204

www.srmup.in

# BONAFIDE CERTIFICATE

Registration no:

*Certified to be the bonafide record of work done by of 5th semester 3rd year B.Tech degree course in SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, NCR Campus of Department of Computer Science & Engineering, in COMPILER DESIGN (18CSC304J), during the academic year 2023-2024 (Even).*

Lab In charge

Head of the Department (CSE)

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
# DELHI-NCR CAMPUS, MODINAGAR

## List of Experiments

**Year/Course & Branch :** IIIrd YEAR / B.TECH & CSE (ALL SPECIALIZATION)

**Subject Name & Code :** COMPILER DESIGN (18CSC304J)

| S. No. | Title | Date | Signature |
|---|---|---|---|
| 1 | Implementation of Lexical Analyzer | | |
| 2 | Conversion from Regular Expression to NFA | | |
| 3 | Conversion from NFA to DFA | | |
| 4 | Elimination of Ambiguity, Left Recursion and Left Factoring | | |
| 5 | FIRST AND FOLLOW computation | | |
| 6 | PREDICTIVE PARSING TABLE | | |
| 7 | Shift Reduce Parsing | | |
| 8 | Computation of LEADING AND TRAILING | | |
| 9 | Computation of LR (0) items | | |
| 10 | Intermediate code generation – Postfix, Prefix | | |
| 11 | Intermediate code generation – Quadruple, Triple, Indirect triple | | |
| 12 | A simple code Generator | | |
| 13 | Implementation of DAG | | |

## Value Added Experiments

| S. No. | Title | Date | Signature |
|---|---|---|---|
| 1 | Implementation of Global Data Flow Analysis | | |
| 2 | Implement any one storage allocation strategies (heap, stack, static) | | |

FACULTY NAME & SIGNATURE

LAB INCHARGE NAME & SIGNATURE

HOD (CSE)

# EXPERIMENT 1

## 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. ASCII Range TOKEN TYPE

97-122 Keyword else identifier

48-57 Constant else operator

Greater than 12 Symbol

6. Print the token types.

7. Stop

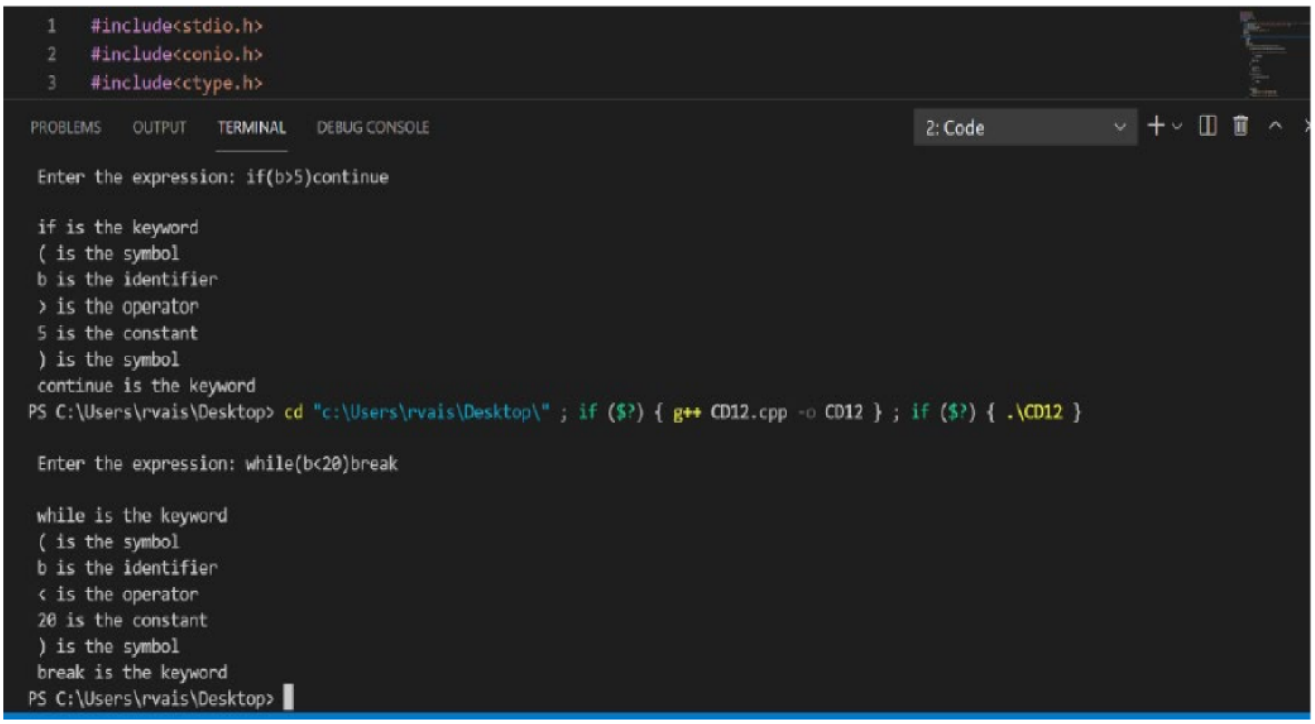
Program (lexi.c): /* Lexical Analyzer */

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
using namespace std;
int 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;
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(strcmp(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
return 0;
}
```

OUTPUT:



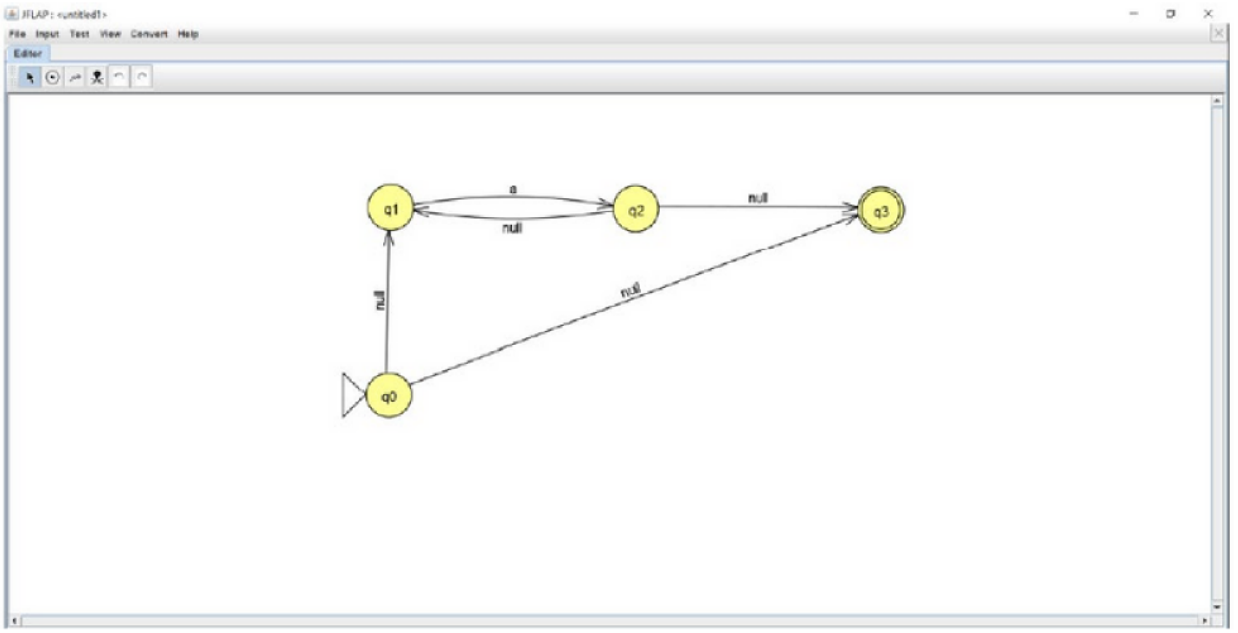

Result: The Program Executed successfully.

# EXPERIMENT 2

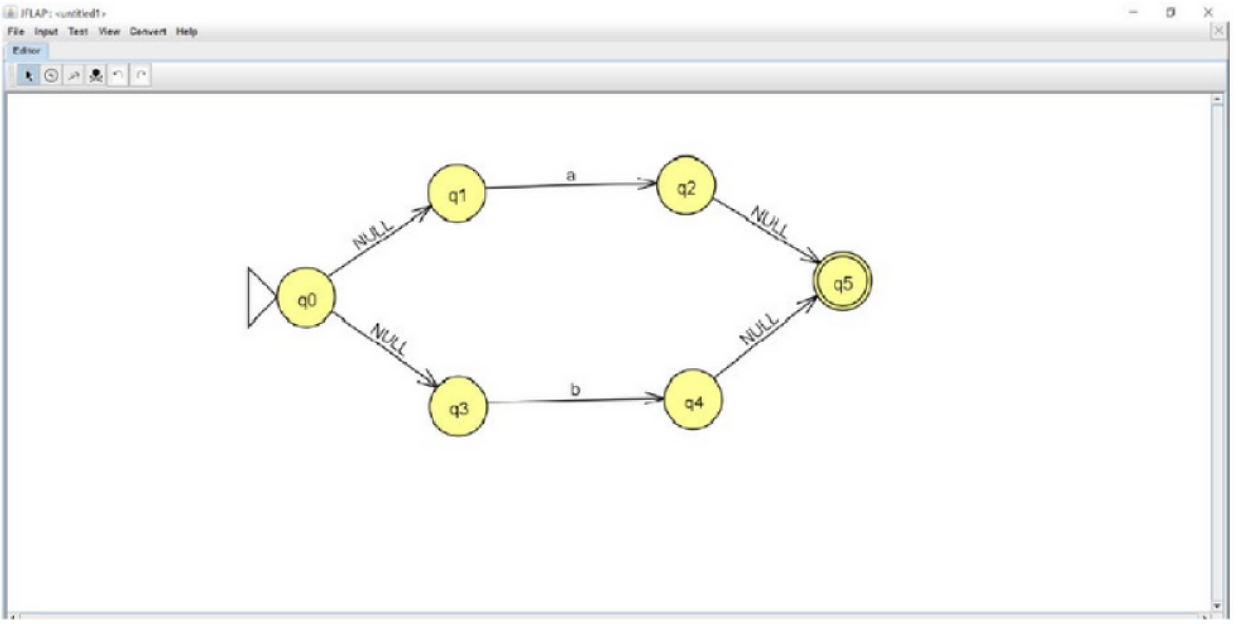
## Conversion from Regular Expression to NFA

Aim: To convert the given Regular expression to NFA by using JFLAP. 1.

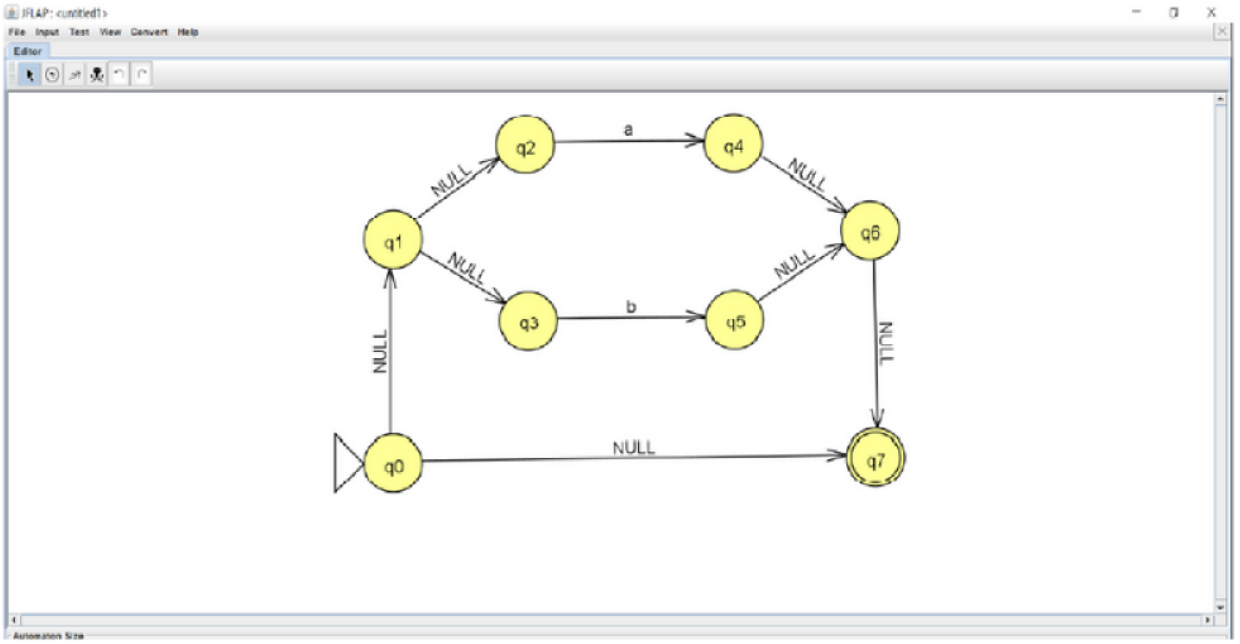
a*



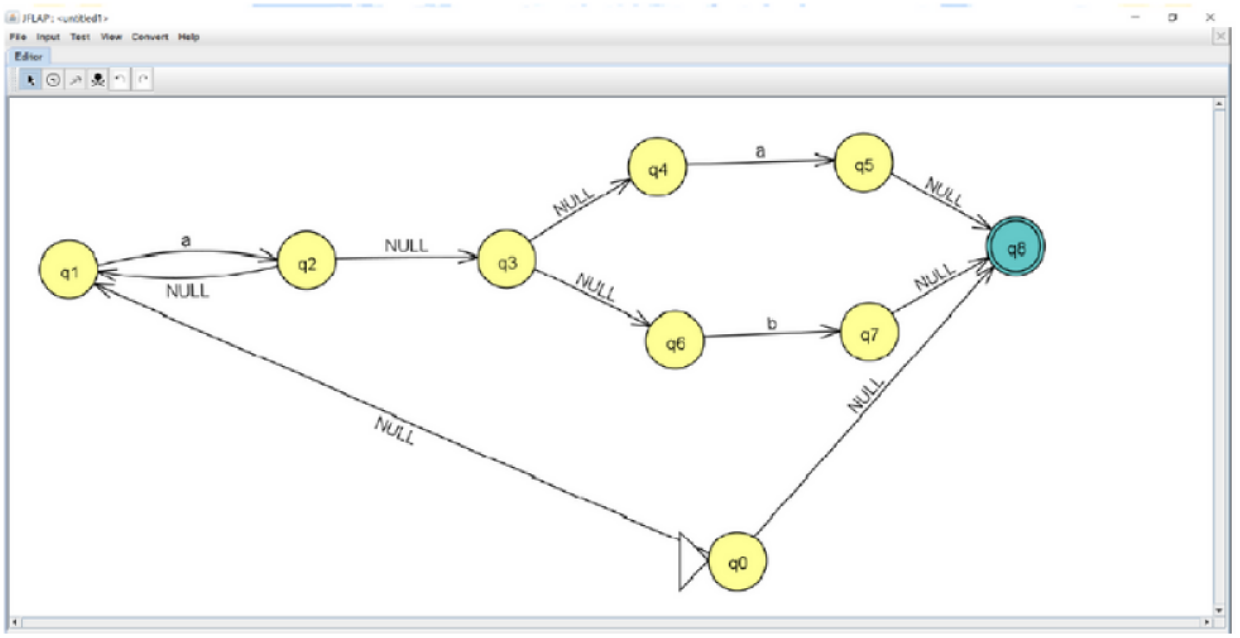

2. (a+b)



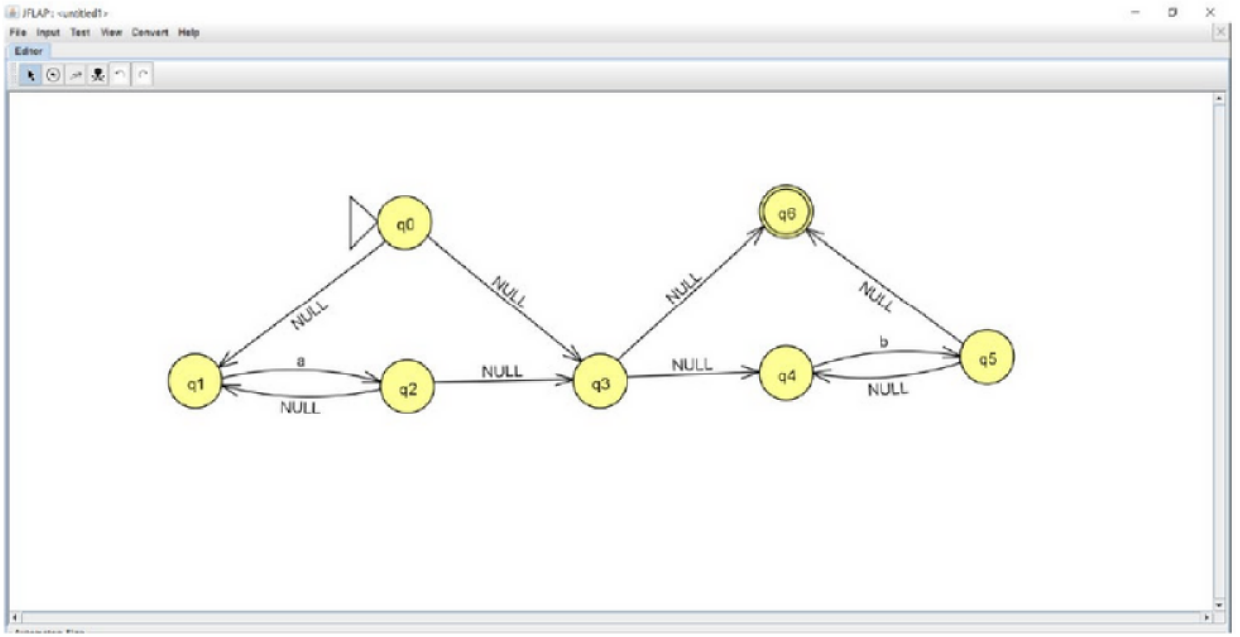

3. (a+b)*


JFLAP: <untitled1>
File Input Test View Convert Help
Editor
q0
q1
q2
q3
q4
q5
q6
q7
a
b
NULL

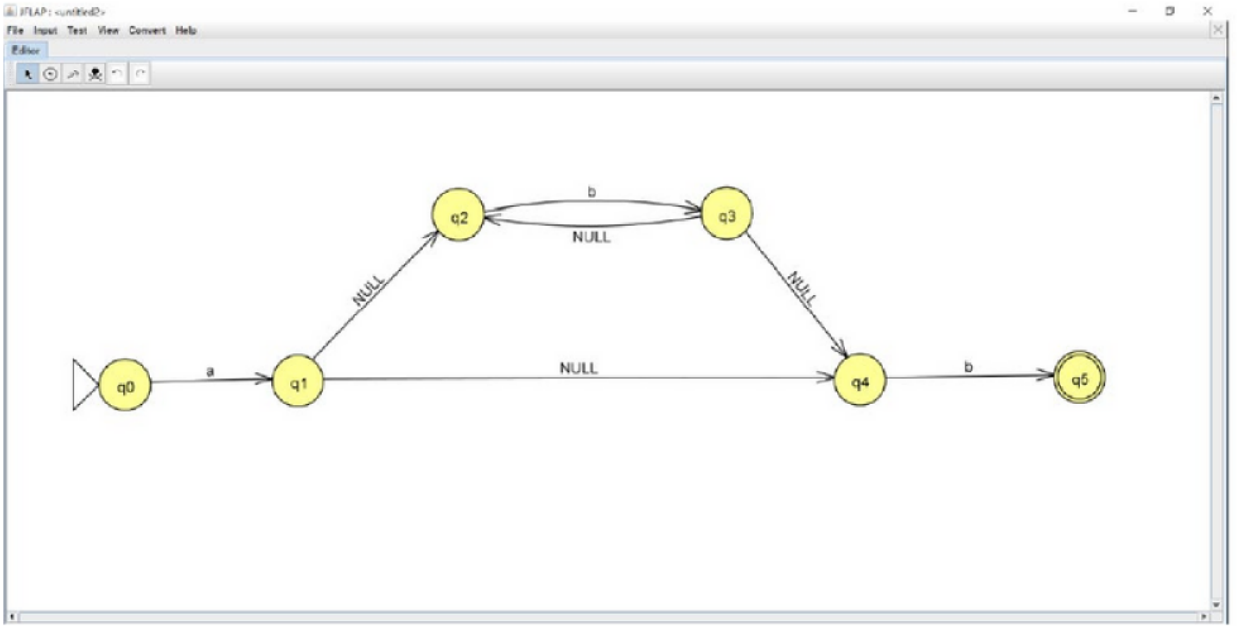
4. a*(a+b)


JFLAP: <untitled1>
File Input Test View Convert Help
Editor
q0
q1
q2
q3
q4
q5
q6
q7
q8
a
b
NULL

5. a*b*




6. ab*b




Result: We converted the given Regular expression to NFA.

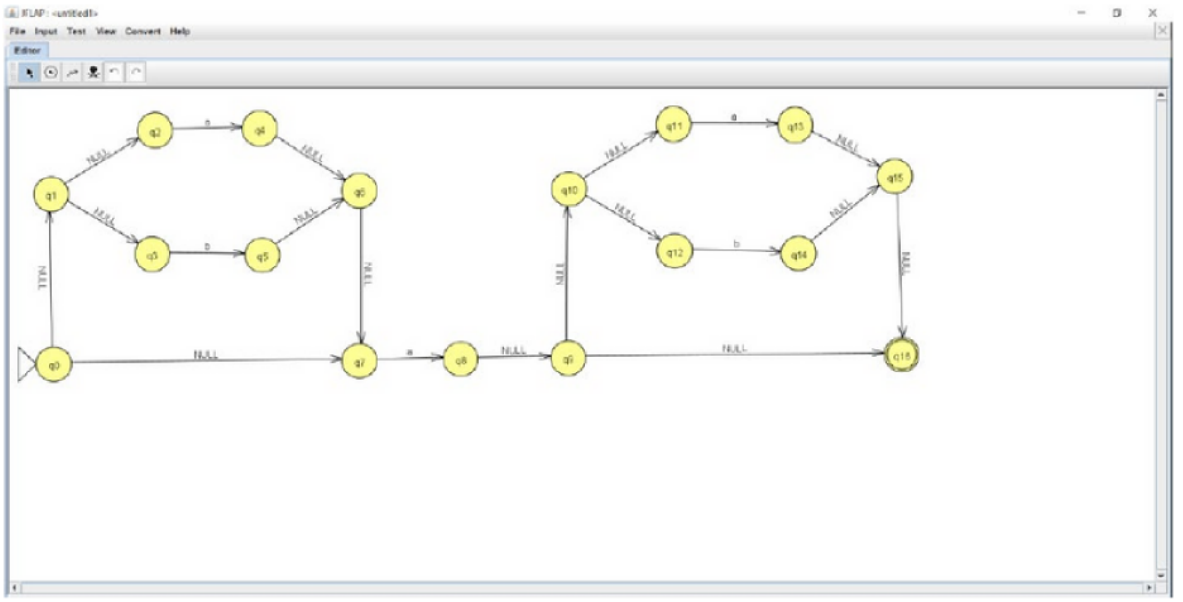
# EXPERIMENT 3

## Conversion from NFA to DFA

Aim: To convert the given Regular expression to DFA by using JFLAP.

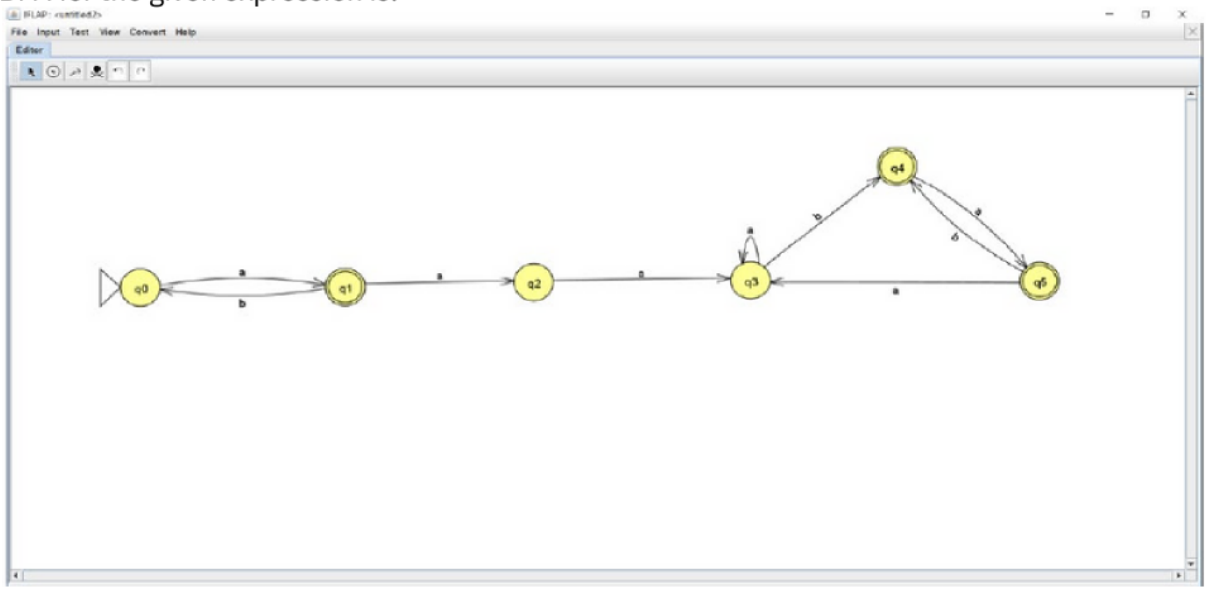
Ques: (a+b) *a(a+b) *

NFA for the given expression is:




OUTPUT: -

DFA for the given expression is:




Result: We converted the given Regular expression to DFA.

# EXPERIMENT 4

## Elimination of Ambiguity, Left Recursion and Left Factoring

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

Program:

Elimination of left recursion:

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

```
cout<<endl;
if (d[k] == '#')
{
b.push_back(d[0]);
b += "\'";
}
else
{
for (j = k; j < k + length[i]; j++)
{
b.push_back(d[j]);
}
k = j + 1;
b.push_back(d[0]);
b += "\'|";
}
}
else
{
cout<<"Production: "<< i + 1 ;
cout<<" has left recursion";
cout<< endl;
if (d[k] != '#')
{
for (l = k + 1; l < k + length[i]; l++) {
a.push_back(d[l]);
}
k = l + 1;
a.push_back(d[0]);
a += "\'|";
}
}
}
a += "#";
cout << b << endl;
cout << a << endl;
return 0;
}
```

Left factoring:

```
#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:

Elimination of left recursion:

```
Enter the Parent Non-Terminal : S
Enter the number of productions : 2
Enter Production 1 : SOS1S
Enter Production 2 : 01
Production Rule : S->SOS1S|01
Production 1 has left recursion.
Production 2 does not have left recursion.
The Grammar after Eliminating the Left-Recursion is
S->01S'|
S'->OS1SS'|#
```

Left Factoring

```
Enter the Parent Non-Terminal : E
Enter total number of productions : 4
Enter the Production 1 : i
Enter the Production 2 : iE
Enter the Production 3 : (E)
Enter the Production 4 : iE+E
The Production Rule is : E->i|iE|(E)|iE+E
After Left Factoring :
E->iE'|(E)
E'->#|EE''
E''->#|+E
```

# EXPERIMENT 5

## FIRST AND FOLLOW computation

Aim: Write a program in C/C++ to find the FIRST and FOLLOW 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 Xi ≠ Xi+1 then

i. Print Result array of Xi which contain FIRST(Xi)

b. If first element of Xi of PArray is Terminal or ε Then

i. Add Result = Result U first element

c. If first element of Xi 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 Xi which contain FIRST(Xi)

6. End

### Procedure searchFirst(i, PArray, N)

1. Repeat steps Loop j=i+1 to N

a. If first element of Xj of PArray is Non-Terminal Then

i. searchFirst(j, of PArray, N)

b. If first element of Xj 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 Xj which contain FIRST(Xj) 4.

End

Algorithm FOLLOW:

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->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).

5. END.

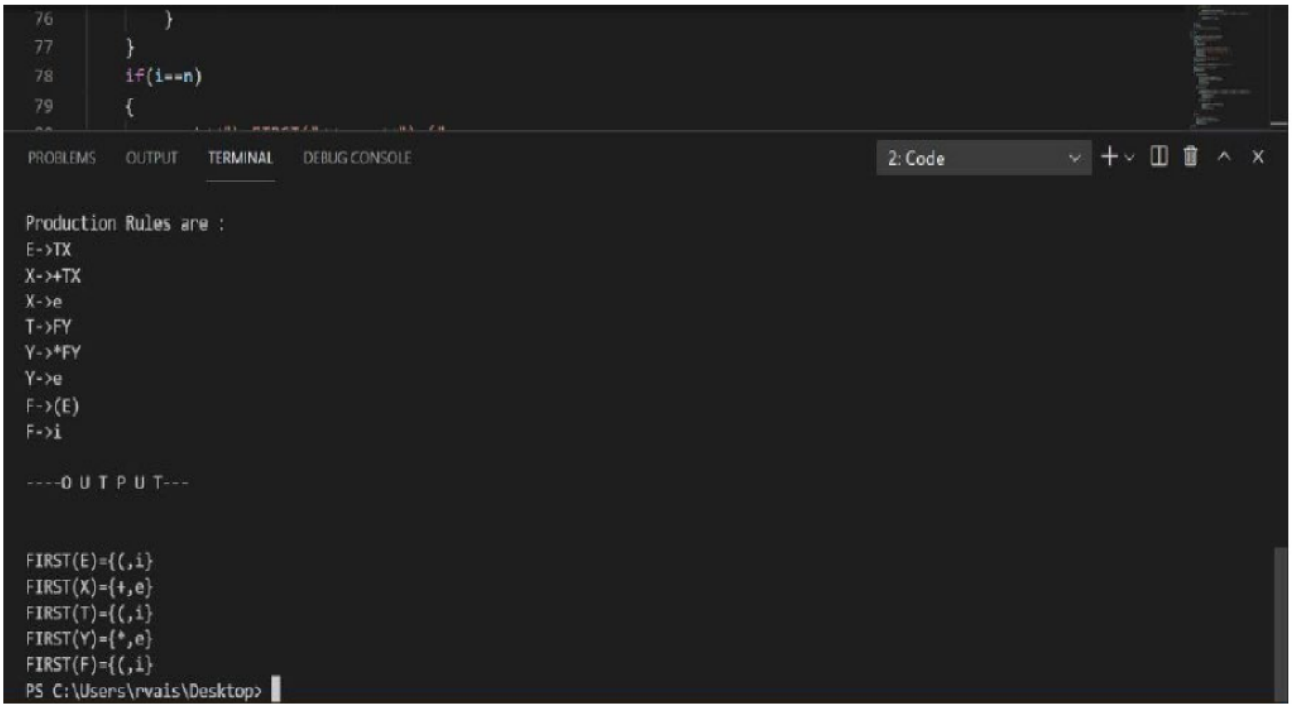
Program: // FIRST

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<ctype.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<<"\nHow 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";//<<";"<<r[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];
//cout<<"\n3";
}
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: -



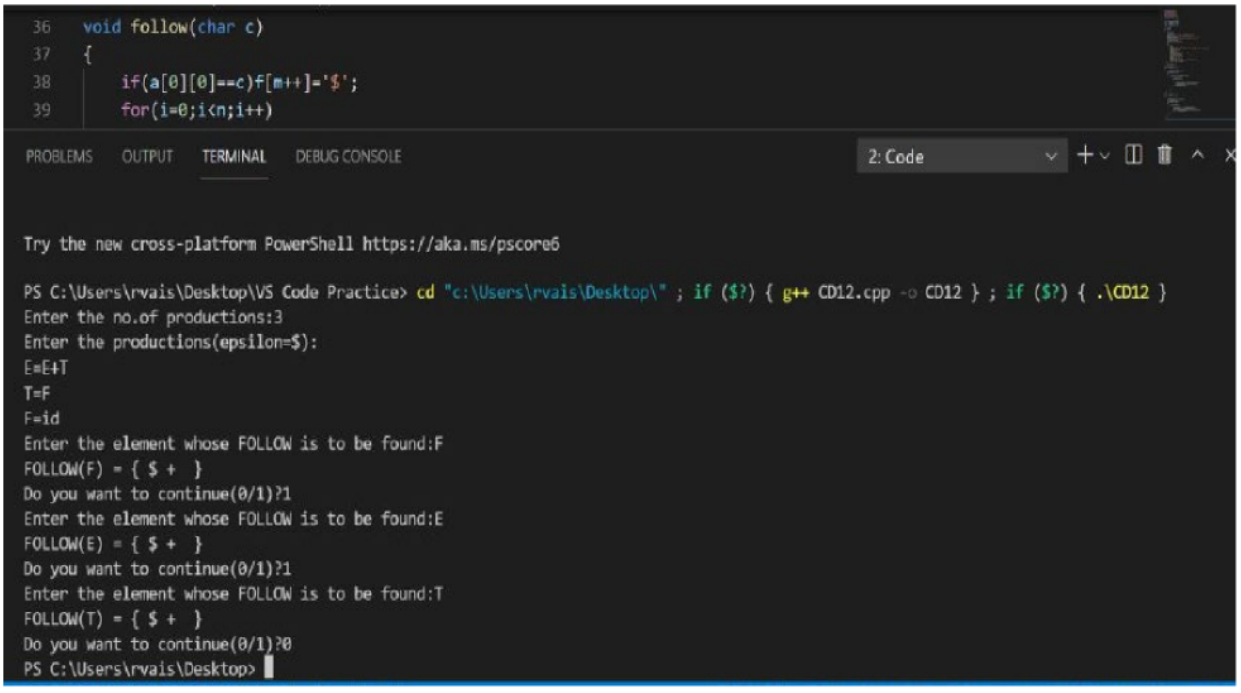

Result: The Program Executed successfully.

Program: //FOLLOW

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
using namespace std;
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=$):\n");
for(i=0;i<n;i++)
scanf("%s%c",a[i],&ch);
do
{
m=0;
printf("Enter the element whose FOLLOW is to be found:");
scanf("%c",&c);
follow(c);
printf("FOLLOW(%c) = { ",c);
for(i=0;i<m;i++)
printf("%c ",f[i]);
printf(" }\n");
printf("Do you want to continue(0/1)?");
scanf("%d%c",&z,&ch);
}
while(z==1);
}
void follow(char c)
{
if(a[0][0]==c)f[m++]='$';
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: -




Result: The Program Executed successfully.

# EXPERIMENT 6

## Predictive Parsing Table

Aim: Write a program in c for construction of predictive parser table.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
using namespace std;
char prol[7][10] ={"S","A","A","B","B","C","C"};
char pror [7][10] ={"A","Bb","Cd","aB","@","Cc","@"};
char prod [7][10] ={"S->A","A->Bb","A->Cd","B->aB","B->@","C->Cc","C->@"}; char first
[7][10] ={"abcd","ab","cd","a@","@","c@","@"}; char
follow [7][10] ={"$","$","$","a$","b$","c$","d$"};
char table [5][6][10];
int numr (char c)
{
switch(c)
{
case 'S': return 0;
case 'A': return 1;
case 'B': return 2;
case 'C': return 3;
case 'a': return 0;
case 'b': return 1;
case 'c': return 2;
case 'd': return 3;
case '$': return 4;
}
return (2);
}
int main ()
{
int i,j,k;
for (i=0; i<5; i++)
for (j=0; j<6; j++)
strcpy(table[i][j]," ");
printf ("\nThe following is the predictive parsing table for the following grammar:\n");
for (i=0; i<7; i++)
printf ("%s\n",prod[i]);
printf ("\nPredictive parsing table is\n");
fflush (stdin);
for (i=0; i<7; i++)
{
```

```
k=strlen(first[i]);
for (j=0; j<10; j++)
if(first[i][j] !='@')
strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);
}
for(i=0;i<7;i++)
{
if(strlen(pror[i])==1)
{
if(pror[i][0]=='@')
{
k=strlen(follow[i]);
for(j=0;j<k;j++)
strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);
}
}
}
strcpy(table[0][0]," ");
strcpy(table[0][1],"a");
strcpy(table[0][2],"b");
strcpy(table[0][3],"c");
strcpy(table[0][4],"d");
strcpy(table[0][5],"$");
strcpy(table[1][0],"S");
strcpy(table[2][0],"A");
strcpy(table[3][0],"B");
strcpy(table[4][0],"C");
printf("\n---------------------------------------------------\n");
for(i=0;i<5;i++)
for(j=0;j<6;j++){
printf("%-10s",table[i][j]);
if(j==5)
printf("\n---------------------------------------------------\n");
}
return 0;
}
```

OUTPUT:


Result: The Program Executed successfully.

# EXPERIMENT 7

## 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.

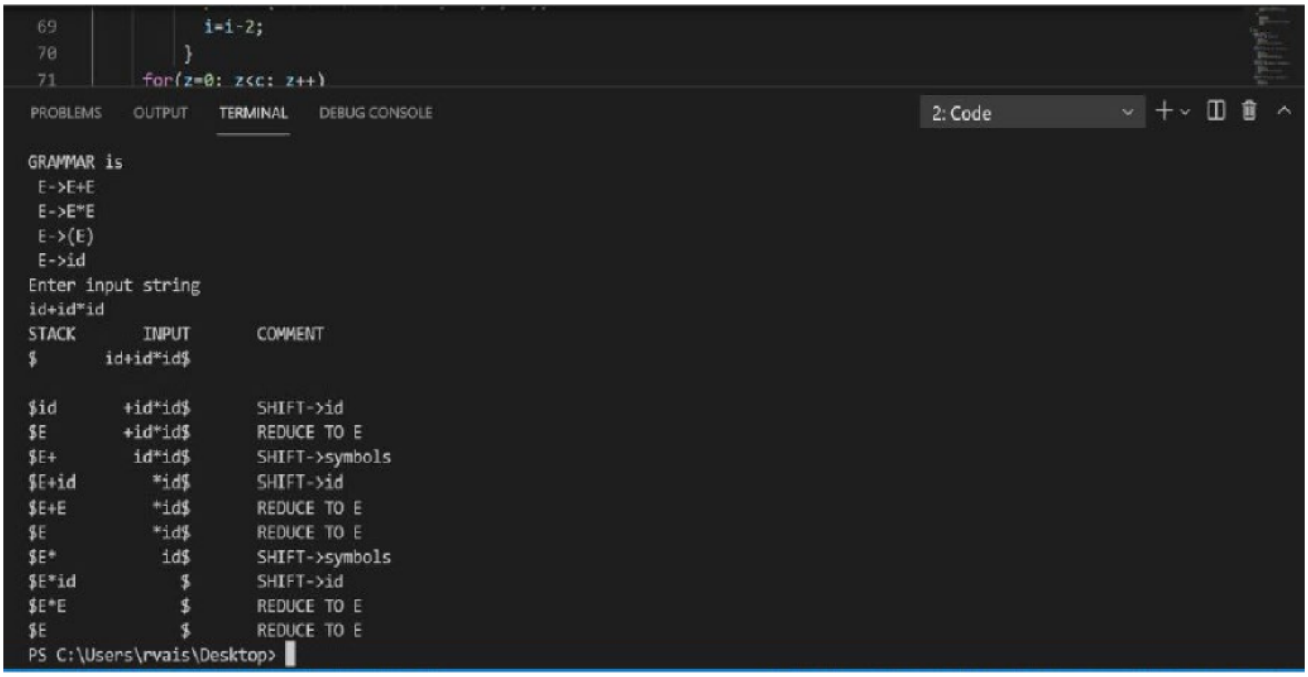
Program (srp.cpp):

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

OUTPUT:




Code for another Grammar: -

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
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; //return to main
}
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 //else reject
printf("Reject\n");
}
```

OUTPUT:

```
GRAMMAR is -
E->2E2
E->3E3
E->4

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


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

Result: The Program Executed successfully.

# EXPERIMENT 8

## Computation of leading and trailing

Aim: Write a program for finding the leading and trailing.

Program:

```
#include<iostream>
#include<string.h>
using namespace std;
int nt,t,top=0;
char   s[50],NT[10],T[10],st[50],l[10][10],tr[50][50];
int searchnt(char a)
{
int count=-1,i;
for(i=0;i<nt;i++)
{
if(NT[i]==a)
return i;
}
return count;
}
int searchter(char a)
{
int count=-1,i;
for(i=0;i<t;i++)
{
if(T[i]==a)
return i;
}
return count;
}
void push(char a)
{
s[top]=a;
top++;
}
char pop()
{
top--;
return s[top];
}
void installl(int a,int b)
{
```

```
if(l[a][b]=='f')
{
l[a][b]='t';
push(T[b]);
push(NT[a]);
}
}
void installt(int a,int b)
{
if(tr[a][b]=='f')
{
tr[a][b]='t';
push(T[b]);
push(NT[a]);
}
}
int main()
{
int i,s,k,j,n;
char pr[30][30],b,c;
cout<<"Enter the no of productions:\n";
cin>>n;
cout<<"Enter the productions one by one\n";
for(i=0;i<n;i++)
cin>>pr[i];
nt=0;
t=0;
for(i=0;i<n;i++)
{
if((searchnt(pr[i][0]))==-1)
NT[nt++]=pr[i][0];
}
for(i=0;i<n;i++)
{
for(j=3;j<strlen(pr[i]);j++)
{
if(searchnt(pr[i][j])==-1)
{
if(searchter(pr[i][j])==-1)
T[t++]=pr[i][j];
}
}
}
for(i=0;i<nt;i++)
{
for(j=0;j<t;j++)
```

```
l[i][j]='f';
}
for(i=0;i<nt;i++)
{
for(j=0;j<t;j++)
tr[i][j]='f';
}
for(i=0;i<nt;i++)
{
for(j=0;j<n;j++)
{
if(NT[(searchnt(pr[j][0]))]==NT[i])
{
if(searchter(pr[j][3])!=-1)
installl(searchnt(pr[j][0]),searchter(pr[j][3]));
else
{
for(k=3;k<strlen(pr[j]);k++)
{
if(searchnt(pr[j][k])==-1)
{
installl(searchnt(pr[j][0]),searchter(pr[j][k]));
break;
}
}
}
}
}
}
while(top!=0)
{
b=pop();
c=pop();
for(s=0;s<n;s++)
{
if(pr[s][3]==b) installl(searchnt(pr[s]
[0]),searchter(c));
}
}
for(i=0;i<nt;i++)
{
cout<<"Leading["<<NT[i]<<"]"<<"\t{";
for(j=0;j<t;j++)
{
if(l[i][j]=='t')
cout<<T[j]<<",";
```

```
}
cout<<"}\n";
}
top=0;
for(i=0;i<nt;i++)
{
for(j=0;j<n;j++)
{
if(NT[searchnt(pr[j][0])]==NT[i])
{
if(searchter(pr[j][strlen(pr[j])-1])!=-1)
installt(searchnt(pr[j][0]),searchter(pr[j][strlen(pr[j])-1]));
else
{
for(k=(strlen(pr[j])-1);k>=3;k--)
{
if(searchnt(pr[j][k])==-1)
{
installt(searchnt(pr[j][0]),searchter(pr[j][k]));
break;
}
}
}
}
}
}
while(top!=0)
{
b=pop();
c=pop();
for(s=0;s<n;s++)
{
if(pr[s][3]==b)
installt(searchnt(pr[s][0]),searchter(c));
}
}
for(i=0;i<nt;i++)
{
cout<<"Trailing["<<NT[i]<<"]"<<"\t{";
for(j=0;j<t;j++)
{
if(tr[i][j]=='t')
cout<<T[j]<<",";
}
cout<<"}\n";
}}
```

OUTPUT:

```
162   while(top!=0)
163   {
164   b=pop();
165   c=pop();
166   for(s=0;s<n;s++)
167   {
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 2: Code

```
Enter the no of productions:
6
Enter the productions one by one
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
Leading[E]      {+,*,(,i,}
Leading[T]      {*,(,i,}
Leading[F]      {(,i,}
Trailing[E]     {+,*,),i,}
Trailing[T]     {*,),i,}
Trailing[F]     {),i,}
PS C:\Users\rvais\Desktop>
```

Result: The Program Executed successfully.

# EXPERIMENT 5

## Computation of LR (0) items

Aim: Write a program in C/C++ to implement LR(0) items.

Algorithm:

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

Program:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
char  prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQR";  int
novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
int noitem=0;
struct Grammar
{
char lhs;
char rhs[8];
}g[20],item[20],clos[20][10];
int isvariable(char variable)
{
for(int i=0;i<novar;i++)
if(g[i].lhs==variable)
return i+1;
return 0;
}
void findclosure(int z, char a)
```

```
{
int n=0,i=0,j=0,k=0,l=0;
for(i=0;i<arr[z];i++)
{
for(j=0;j<strlen(clos[z][i].rhs);j++)
{
if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==a)
{
clos[noitem][n].lhs=clos[z][i].lhs;
strcpy(clos[noitem][n].rhs,clos[z][i].rhs);
char temp=clos[noitem][n].rhs[j]; clos[noitem][n].rhs[j]=clos[noitem]
[n].rhs[j+1]; clos[noitem][n].rhs[j+1]=temp;
n=n+1;
}
}
}
for(i=0;i<n;i++)
{
for(j=0;j<strlen(clos[noitem][i].rhs);j++)
{
if(clos[noitem][i].rhs[j]=='.' && isvariable(clos[noitem][i].rhs[j+1])>0) {
for(k=0;k<novar;k++)
{
if(clos[noitem][i].rhs[j+1]==clos[0][k].lhs)
{
for(l=0;l<n;l++)
if(clos[noitem][l].lhs==clos[0][k].lhs && strcmp(clos[noitem]
[l].rhs,clos[0][k].rhs)==0)
break;
if(l==n)
{
clos[noitem][n].lhs=clos[0][k].lhs; strcpy(clos[noitem][n].rhs,clos[0]
[k].rhs);
n=n+1;
}
}
}
}
}
}
arr[noitem]=n;
int flag=0;
for(i=0;i<noitem;i++)
{
```

```
if(arr[i]==n)
{
for(j=0;j<arr[i];j++)
{
int c=0;
for(k=0;k<arr[i];k++)
if(clos[noitem][k].lhs==clos[i][k].lhs && strcmp(clos[noitem][k].rhs,clos[i]
[k].rhs)==0)
c=c+1;
if(c==arr[i])
{
flag=1;
goto exit;
}
}
}
}
exit:;
if(flag==0)
arr[noitem++]=n;
}
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 augumented 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: -

Result: The Program Executed successfully.

Program: //FOLLOW

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
using namespace std;
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);
}
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: -

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

A->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
 THE SET OF ITEMS ARE
 I0
A->.E
E->.E+T
E->.T
T->.T*F
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).
```

Result: The Program Executed successfully.

# EXPERIMENT 10

## Intermediate code generation – Postfix, Prefix

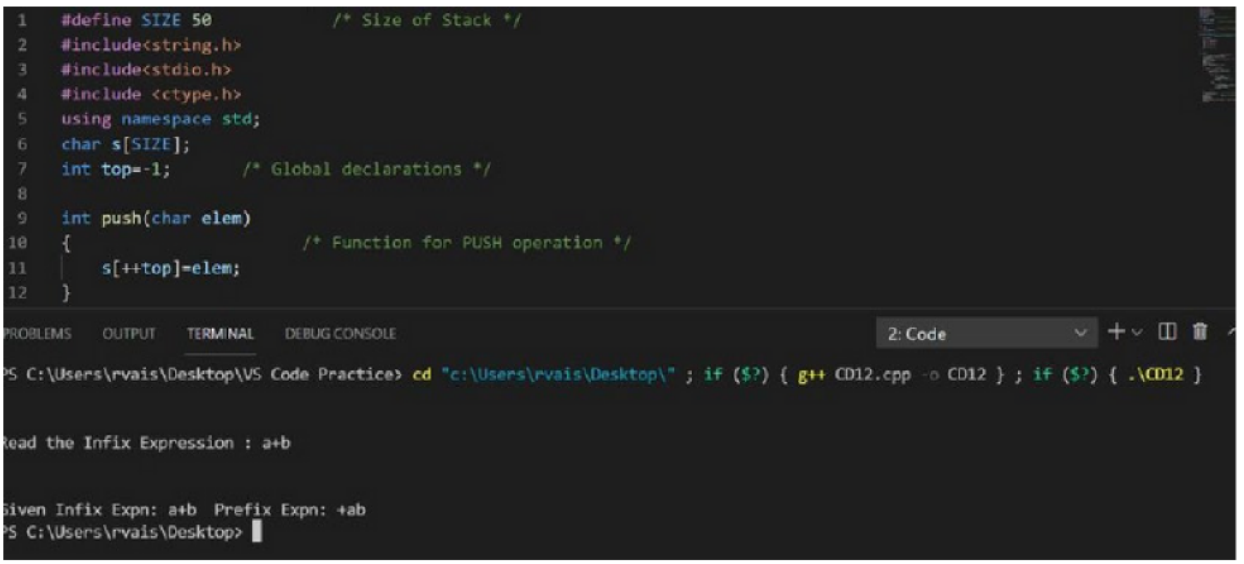
Aim: Write a program in C/C++ or Java to generate Intermediate Code (Prefix and Postfix Expression) from given syntax tree.

Program: //Prefix

```
#define SIZE 50 /* Size of Stack */
#include<string.h>
#include<stdio.h>
#include <ctype.h>
using namespace std;
char s[SIZE];
int top=-1; /* Global declarations */
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("\n\nRead the Infix Expression : ");
scanf("%s",infx);
push('#');
strrev(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 Expn: %s Prefix Expn: %s\n",infx,prfx);
return 0;
}
```

OUTPUT:



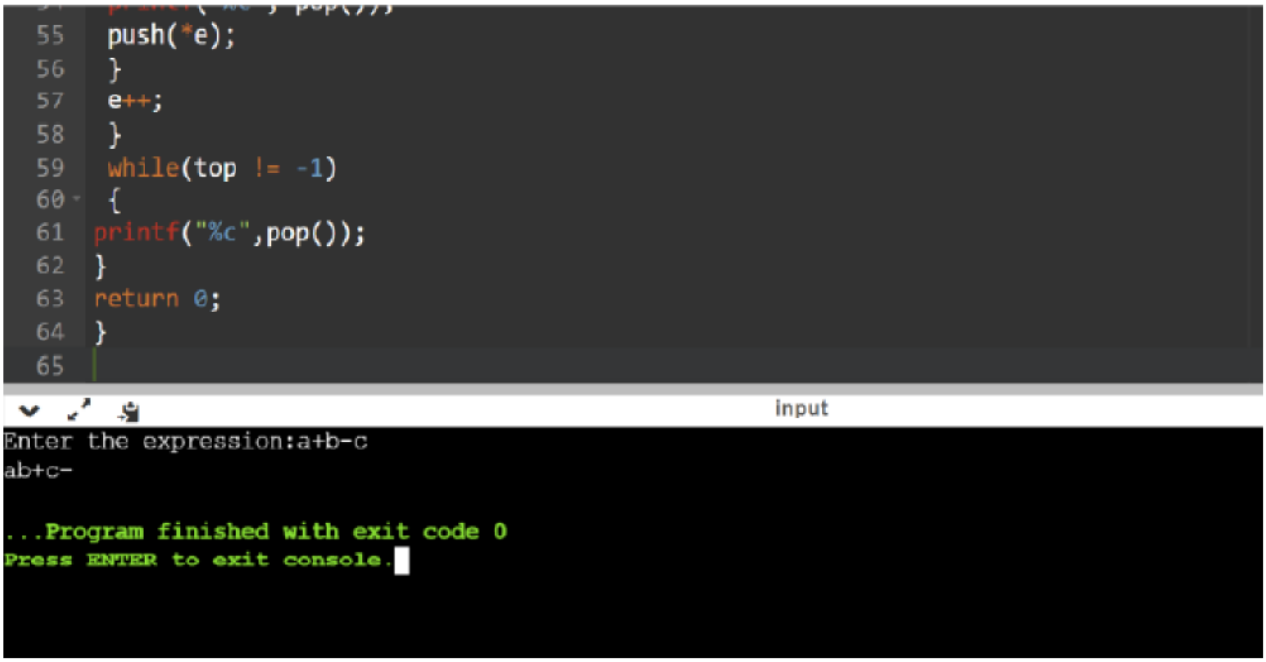

Result: The Program Executed successfully

Program: // Postfix

```
#include<string.h>
#include <stdio.h>
#include <ctype.h>
using namespace std;
char stack[20];
int top=-1;
void push(char x)
{
stack[++top]=x;
}
char pop()
{
if(top==-1)
{
return -1;
}
else
{
return stack[top--];
}
}
//Check the priority of the operator.
int priority(char x)
{
if(x == '(')
return 0;
if(x == '+' || x == '-')
return 1;
if(x == '*' || x == '/')
return 2;
}
int main()
{
char exp[20];
char *e , x;
printf("Enter the expression:");
scanf("%s",exp);
e = exp ;
while(*e != '\0')
{
if(isalnum(*e))
printf("%c",*e);
else if(*e == '(')
push(*e);
else if(*e == ')' )
{
```

```
while(( x =pop() ) != '(' )
printf("%c:",x);
}
else
{
//check greater priority operator.
while(priority(stack[top]) >= priority(*e) )
printf("%c", pop());
push(*e);
}
e++;
}
while(top != -1)
{
printf("%c",pop());
}
return 0;
}
```

OUTPUT:




Result: The Program Executed successfully

# EXPERIMENT 11

## Intermediate code generation – Quadruple, Triple, Indirect triple

Aim: Write a program in C/C++ to implement intermediate code generation - Quadruple, Triple, Indirect triple.

### Algorithm:

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

1. Invoke a function getreg to find out the location L where the result of computation b 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 MOV y' , L to place a copy of y in L.
3. Generate the instruction OP z' , L where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptors.
4. If the current value of y or z have no next uses or not live on exit from the block or in register then alter the register descriptor to indicate that after execution of x : = y op z those register will no longer contain y or z.

### Program:

```
#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]))
{
```

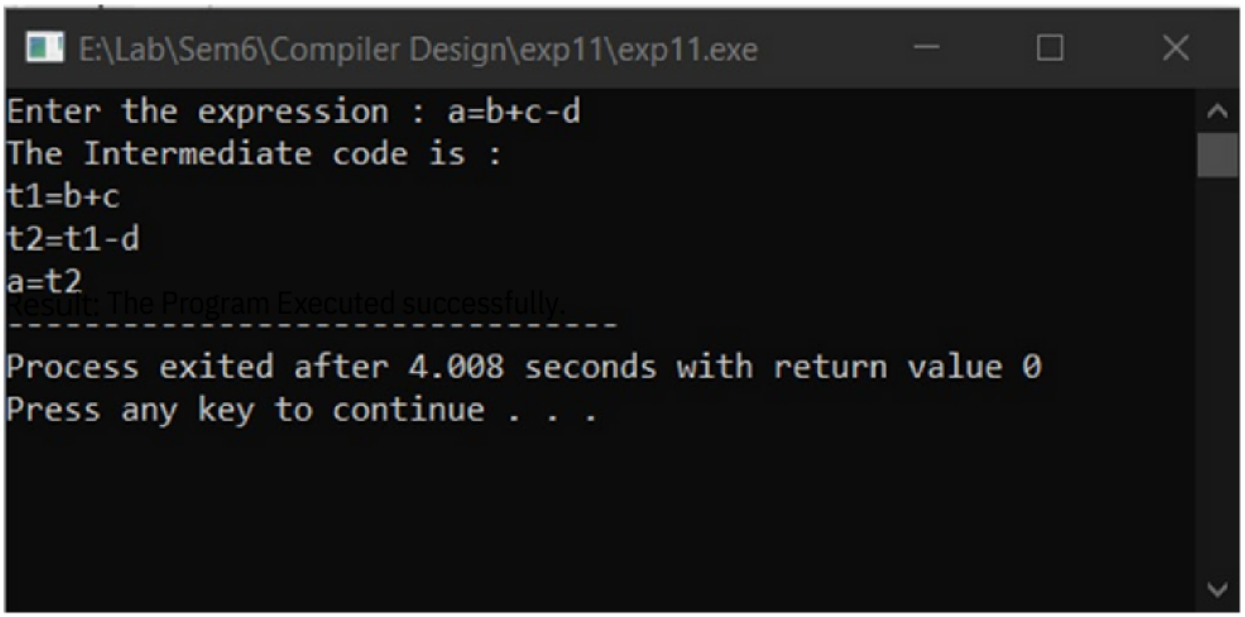
```
 a[0]=j[i-1];
b[0]=j[i+1];
}
if(isdigit(j[i+2]))
{
a[0]=j[i-1];
b[0]='t';
b[1]=j[i+2];
}
if(isdigit(j[i-2]))
{
b[0]=j[i+1];
a[0]='t';
a[1]=j[i-2];
b[1]='\0';
}
if(isdigit(j[i+2]) &&isdigit(j[i-2]))
{
a[0]='t';
b[0]='t';
a[1]=j[i-2];
b[1]=j[i+2];
sprintf(ch,"%d",c);
j[i+2]=j[i-2]=ch[0];
}
if(j[i]=='*')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




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

# EXPERIMENT 12

## Implementation of Code Generator

Aim: To write a C program to implement Simple Code Generator.

Algorithm:

Input: Set of three address code sequence.

Output: Assembly code sequence for three address codes (opd1=opd2, op, opd3).

1- Start
2- Get address code sequence.
3- Determine current location of 3 using address (for 1st operand).
4- If current location not already exist generate move (B,O).
5- Update address of A(for 2nd operand).
6- If current value of B and () is null,exist.
7- If they generate operator () A,3 ADPR.
8- Store the move instruction in memory 9-
9-Stop.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#include<graphics.h>
typedef struct
{
char var[10];
int alive;
}regist;
regist preg[10];
void substring(char exp[],int st,int end)
{
int i,j=0;
char dup[10]="";
for(i=st;i<end;i++)
dup[j++]=exp[i];
dup[j]='0';
strcpy(exp,dup);
}
int getregister(char var[])
{
int i;
for(i=0;i<10;i++)
{
if(preg[i].alive==0)
{
```

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

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

Output:

```
Enter the Three Address Code:
a=b+c
c=a*c
exit
The Equivalent Assembly Code is:
Mov R0,b
Add c,R0
Mov a,R0
Mov R1,a
Mul c,R1
Mov c,R1
```

Result: The Program Executed successfully.

# EXPERIMENT 13

## Implementation of DAG

Aim: Write a c or c++ or java to implement 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:

```cpp
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string exp;
    cout<<"Enter the expression: ";
```

input

```
Enter the expression: a=b+c-5
0->5
1->c
2->b
3->-01
4->+23
5->a
6->=54


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

Result: The Program Executed successfully.

# VALUE ADDED-1

## Implementation of Global Data Flow Analysis

Aim: Write a program in c to implement Global Data Flow Analysis.

Program:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
void input();
void output();
void change(int p,int q,char *res);
void constant();
void expression();
struct expr
{
char op[2],op1[5],op2[5],res[5];
int flag;
}arr[10];
int n;
int main()
{
int ch=0;
input();
constant();
expression();
output();
}
void input()
{
int i;
printf("\n\nEnter the maximum number of expressions:");
scanf("%d",&n);
printf("\nEnter the input : \n");
for(i=0;i<n;i++)
{
scanf("%s",arr[i].op);
scanf("%s",arr[i].op1);
scanf("%s",arr[i].op2);
scanf("%s",arr[i].res);
arr[i].flag=0;
}
}
void constant()
{
int i;
int op1,op2,res;
char op,res1[5];
for(i=0;i<n;i++)
{
if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]))
{
op1=atoi(arr[i].op1);
op2=atoi(arr[i].op2);
op=arr[i].op[0];
```

```
switch(op)
{
case '+':
res=op1+op2;
break;
case '-':
res=op1-op2;
break;
case '*':
res=op1*op2;
break;
case '/':
res=op1/op2;
break;
}
sprintf(res1,"%d",res);
arr[i].flag=1;
change(i,i,res1);
}
}
}
void expression()
{
int i,j;
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(strcmp(arr[i].op,arr[j].op)==0)
{
if(strcmp(arr[i].op,"+")==0||strcmp(arr[i].op,"*")==0)
{
if(strcmp(arr[i].op1,arr[j].op1)==0&&strcmp(arr[i].op2,arr[j].op2)==0    ||
strcmp(arr[i].op1,arr[j].op2)==0&&strcmp(arr[i].op2,arr[j].op1)==0)
{
arr[j].flag=1;
change(i,j,NULL);
}
}
else
{
if(strcmp(arr[i].op1,arr[j].op1)==0&&strcmp(arr[i].op2,arr[j].op2)==0) {
arr[j].flag=1;
change(i,j,NULL);
} }
{ }
{ }
} }
void output()
{
int i=0;
printf("\nOptimized code is : ");
for(i=0;i<n;i++)
{
if(!arr[i].flag)
{
printf("\n%s %s %s %s\n",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
}
```

```
}
}
void change(int p,int q,char *res)
{
int i;
for(i=q+1;i<n;i++)
{
if(strcmp(arr[q].res,arr[i].op1)==0)
if(res == NULL)
strcpy(arr[i].op1,arr[p].res);
else
strcpy(arr[i].op1,res);
else if(strcmp(arr[q].res,arr[i].op2)==0)
if(res == NULL)
strcpy(arr[i].op2,arr[p].res);
else
strcpy(arr[i].op2,res);
}
}
```

OUTPUT:

Enter the maximum number of expressions: 5

Enter the input :

```
+ 4 2 t1
+ a t1 t2
- b a t3
+ a 6 t4
+ t3 t2 t4
```

Optimized code is :

```
+ a 6 t2
- b a t3
+ t3 t2 t4
```

Result: The Program Executed successfully.

VALUE ADDED-2

## Implement any one storage allocation strategies (heap, stack, static)

Aim: Write a program in c to implement Stack storage allocation strategies.

Algorithm:

Step1: Initially check whether the stack is empty
Step2: Insert an element into the stack using push operation
Step3: Insert more elements onto the stack until stack becomes full Step4: Delete an element from the stack using pop operation
Step5: Display the elements in the stack
Step6: Top the stack element will be displayed

Program:

```
//implementation of heap allocation storage strategies//
#include<stdio.h>
#include<stdlib.h>
#define TRUE 1
#define FALSE 0
typedef struct Heap
{
int data;
struct Heap *next;
}
node;
node *create();
void main()
{
int choice,val;
char ans;
node *head;
void display(node *);
node *search(node *,int);
node *insert(node *);
void dele(node **);
head=NULL;
do
{
printf("\nprogram to perform various operations on heap using dynamic memory management");
printf("\n1.create");
printf("\n2.display");
printf("\n3.insert an element in a list");
printf("\n4.delete an element from list");
printf("\n5.quit");
printf("\nenter your chioce(1-5)");
```

```
scanf("%d",&choice);
switch(choice)
{
case 1:head=create();
break;
case 2:display(head);
break;
case 3:head=insert(head);
break;
case 4:dele(&head);
break;
case 5:exit(0);
default:
printf("invalid choice,try again");
}
}
while(choice!=5);
}
node* create()
{
node *temp,*New,*head;
int val,flag;
char ans='y';
node *get_node();
temp=NULL;
flag=TRUE;
do
{
printf("\n enter the element:");
scanf("%d",&val);
New=get_node();
if(New==NULL)
printf("\nmemory is not allocated");
New->data=val;
if(flag==TRUE)
{
head=New;
temp=head;
flag=FALSE;
}
else
{
temp->next=New;
temp=New;
}
printf("\ndo you want to enter more elements?(y/n)");
```

```
}
while(ans=='y');
printf("\nthe list is created\n");
return head;
}
node *get_node()
{
node *temp; temp=
(node*)malloc(sizeof(node)); temp-
>next=NULL;
return temp;
}
void display(node *head)
{
node *temp;
temp=head;
if(temp==NULL)
{
printf("\nthe list is empty\n");
return;
}
while(temp!=NULL)
{
printf("%d->",temp->data); temp=temp-
>next;
}
printf("NULL");
}
node *search(node *head,int key)
{
node *temp;
int found;
temp=head;
if(temp==NULL)
{
printf("the linked list is empty\n");
return NULL;
}
found=FALSE;
while(temp!=NULL && found==FALSE) {
if(temp->data!=key)
temp=temp->next;
else
found=TRUE;
}
```

```
if(found==TRUE)
{
printf("\nthe element is present in the list\n");
return temp;
}
else
{
printf("the element is not present in the list\n");
return NULL;
}
}
node *insert(node *head)
{
int choice;
node *insert_head(node *);
void insert_after(node *);
void insert_last(node *);
printf("n1.insert a node as a head node");
printf("n2.insert a node as a head node");
printf("n3.insert a node at intermediate position in t6he list");
printf("\nenter your choice for insertion of node:");
scanf("%d",&choice);
switch(choice)
{
case 1:head=insert_head(head);
break;
case 2:insert_last(head);
break;
case 3:insert_after(head);
break;
}
return head;
}
node *insert_head(node *head)
{
node *New,*temp;
New=get_node();
printf("\nEnter the element which you want to insert");
scanf("%d",&New->data);
if(head==NULL)
head=New;
else
{
temp=head;
New->next=temp;
head=New;
```

```
}
return head;
}
void insert_last(node *head)
{
node *New,*temp;
New=get_node();
printf("\nenter the element which you want to insert");
scanf("%d",&New->data);
if(head==NULL)
head=New;
else
{
temp=head;
while(temp->next!=NULL)
temp=temp->next;
temp->next=New;
New->next=NULL;
}
}
void insert_after(node *head)
{
int key;
node *New,*temp;
New=get_node();
printf("\nenter the elements which you want to insert");
scanf("%d",&New->data);
if(head==NULL)
{
head=New;
}
else
{
printf("\enter the element which you want to insert the node");
scanf("%d",&key);
temp=head;
do
{
if(temp->data==key)
{
New->next-temp->next;
temp->next=New;
return;
}
else
temp=temp->next;
```

```
}
while(temp!=NULL);
}
}
node *get_prev(node *head,int val)
{
node *temp,*prev;
int flag;
temp=head;
if(temp==NULL)
return NULL;
flag=FALSE;
prev=NULL;
while(temp!=NULL && ! flag)
{
if(temp->data!=val)
{
prev=temp;
temp=temp->next;
}
else
flag=TRUE;
}
if(flag)
return prev;
else
return NULL;
}
void dele(node **head)
{
node *temp,*prev;
int key;
temp=*head;
if(temp==NULL)
{
printf("\nthe list is empty\n");
return;
}
printf("\nenter the element you want to
delete:"); scanf("%d",&key);
temp=search(*head,key);
if(temp!=NULL)
{
prev=get_prev(*head,key);
if(prev!=NULL)
{
```

```
prev->next=temp->next;
free(temp);
}
else
{
*head=temp->next;
free(temp);
}
printf("\nthe element is deleted\n");

}
}
```

Output:

```
program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)2

the list is empty

program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)5
```

Result: The Program Executed successfully.