DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR: 2020-21 EVEN SEMESTER

Lab Manual:-

| | | |
|---|---|---|
| Programme(UG/PG) | : | UG |
| Semester | : | VI |
| Course Code | : | 18CSC304J. |
| Course Title | : | COMPILER DESIGN |
| FACULTY | : | DR. BALAMURUGAN P |

DONE BY:-


| | | |
|---|---|---|
| Name | : | A.Venkata DineshReddy |
| Reg-No | : | RA1811028010098. |
| SEC | : | J1 |



SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
*(Deemed to be* **University** *u/s 3 of UGC Act, 1956)*

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM IST **(Under section 3 of UGC Act, 1956) SRM , Kattankulathur- 603203 Kancheepuram District**

# INDEX:-

A Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 1
Date : 11-01-2022

# Implementation of Lexical Analyzer

**Aim :-** To write a program implementing a lexical analyzer

**Algorithm :-**

1) Start
2) Get the input program from the file prog.txt
3) Read the program line by line and checks if each word in a line is a keyword, identifier, constant or an operator.
4) If the word read is an identifier, assign a number to the identifier and make an entry into the symbol table stored in symbols related file.
5) For each lexeme read & generate a token as follows:
   a) If the lexeme is an identifier, then the token generated is of the form <id, number>
   b) If the lexeme is an operator, then the token generated is of the form <op, operator>
   c) If the lexeme is a constant, then the token generated is <const, value>
   d) If the lexeme is a keyword, then the token is the keyword itself.
6) The stream of tokens generated are displayed in the console output.
7) Stop

## Program:-

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

int isKeyword(char buffer[]){
char keywords[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 i, flag = 0;
for(i = 0; i < 32; ++i){
if(strcmp(keywords[i], buffer) == 0){
flag = 1;
break;
}
}
return flag;
}

int main(){
char ch, buffer[15], operators[] = "+-*/%=";
FILE *fp;
int i,j=0;
fp = fopen("E:/program.txt","r");
if(fp == NULL){
printf("error while opening the file\n");
exit(0);
}
while((ch = fgetc(fp)) != EOF){
   for(i = 0; i < 6; ++i){
   if(ch == operators[i])
   printf("%c is operator\n", ch);
   }

   if(isalnum(ch)){
```

```
  buffer[j++] = ch;
  }
  else if((ch == ' ' || ch == '\n') && (j != 0)){
  buffer[j] = '\0';
  j = 0;

  if(isKeyword(buffer) == 1)
  printf("%s is keyword\n", buffer);
  else
  printf("%s is indentifier\n", buffer);
  }

}
fclose(fp);
return 0;
}
```

## Program.txt:-

```
void main()

{

int a,b,c;

c=a+b;

}
```

## Output:-


## Result:-

The implementation of lexical analyzer in c program was compiled and executed successfully.

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 2
Date: 28-01-2022

# Conversion from Regular Expression to NFA

**Aim** :- To write a program for converting Regular Expression to NFA.

**Procedure:-**

1) Start
2) Get the input from the user
3) Initialize seperate variables and functions for postfix
4) Create seperate methods for different operators like +, *, -
5) For '-' operator. Initialize a seperate method by using varius stack functions.
6) Continue the same method for the other operators also
7) Regular expression is in the form like a.b (or) a+b
8) Display the output
9) Stop

## Program:-

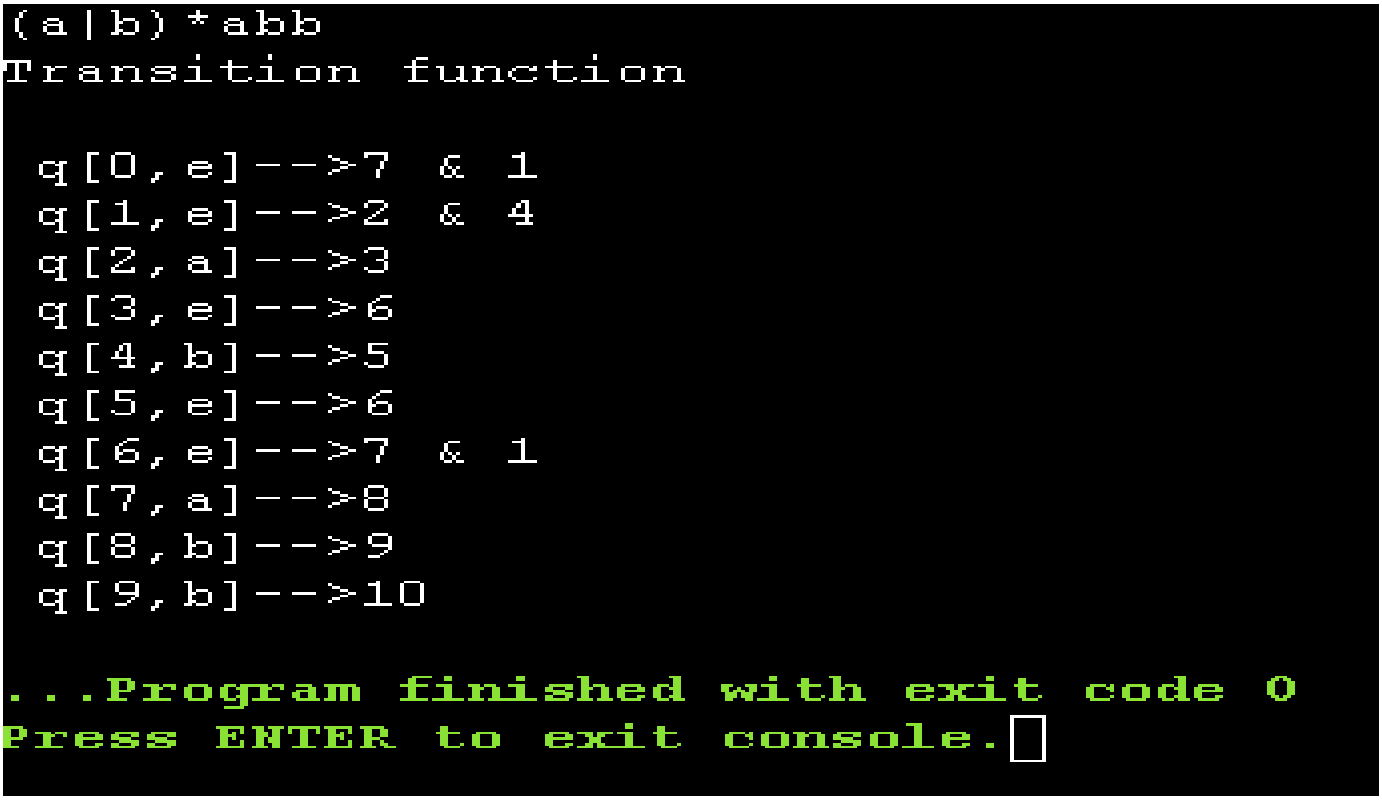
```
#include<stdio.h>
#include<string.h>
int main()
{
	char reg[20];
	int q[20][3],i,j,len,a,b;
	for(a=0;a<20;a++)
	{
		for(b=0;b<3;b++)
		{
			q[a][b]=0;
		}
	}
	scanf("%s",reg);
	len=strlen(reg);
	i=0;
j=1;
	while(i<len)
	{
		if(reg[i]=='a'&&reg[i+1]!='|'&&reg[i+1]!='*')
		{
			q[j][0]=j+1;
			j++;
		}
		if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='*')
		{
			q[j][1]=j+1;
```

```
        j++;
}
if(reg[i]=='e'&&reg[i+1]!='|'&&reg[i+1]!='*')
{
        q[j][2]=j+1;
        j++;
}
if(reg[i]=='a'&&reg[i+1]=='|'&&reg[i+2]=='b')
{
        q[j][2]=((j+1)*10)+(j+3);
        j++;
        q[j][0]=j+1;
        j++;
        q[j][2]=j+3;
        j++;
        q[j][1]=j+1;
        j++;
        q[j][2]=j+1;
        j++;
        i=i+2;
}
if(reg[i]=='b'&&reg[i+1]=='|'&&reg[i+2]=='a')
{
        q[j][2]=((j+1)*10)+(j+3);
        j++;
        q[j][1]=j+1;
        j++;
        q[j][2]=j+3;
```

```
        j++;
        q[j][0]=j+1;
        j++;
        q[j][2]=j+1;
        j++;
        i=i+2;
}
if(reg[i]=='a'&&reg[i+1]=='*')
{
        q[j][2]=((j+1)*10)+(j+3);
        j++;
        q[j][0]=j+1;
        j++;
        q[j][2]=((j+1)*10)+(j-1);
        j++;
}
if(reg[i]=='b'&&reg[i+1]=='*')
{
        q[j][2]=((j+1)*10)+(j+3);
        j++;
        q[j][1]=j+1;
        j++;
        q[j][2]=((j+1)*10)+(j-1);
        j++;
}
if(reg[i]==')'&&reg[i+1]=='*')
{
        q[0][2]=((j+1)*10)+1;
```

```
                q[j][2]=((j+1)*10)+1;
                j++;
        }
        i++;
    }
    printf("Transition function \n");
    for(i=0;i<=j;i++)
    {
        if(q[i][0]!=0)
            printf("\n q[%d,a]-->%d",i,q[i][0]);
        if(q[i][1]!=0)
            printf("\n q[%d,b]-->%d",i,q[i][1]);
        if(q[i][2]!=0)
        {
            if(q[i][2]<10)
                printf("\n q[%d,e]-->%d",i,q[i][2]);
            else
                printf("\n q[%d,e]-->%d & %d",i,q[i][2]/10,q[i][2]%10);
        }
    }
    return 0;
}
```

## Output:-




Result:-

The program to convert regular expressions to NFA was implemented successfully.

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 3

Date : 10-02-2022

# Conversion of NFA to DFA

**Aim :-** To write a program for converting NFA to DFA

**Procedure:-**

1) Start
2) Get the input from user
3) Implement the conversion of Nfa to Dfa using the logic
4) Print the values
5) Stop the program.

## Program:-

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

int main()
{
   vector<vector<int>> nfa( 5 , vector<int> (3));
   vector<vector<int>> dfa( 10 , vector<int> (3));
   for(int i=1;i<5;i++){
      for(int j=1;j<=2;j++){
         int h;
         if (j == 1){
            cout << "nfa [" << i << ", a]: ";
         }
         else{
            cout << "nfa [" << i << ", b]: ";
         }
         cin>>h;
         nfa[i][j]=h;
      }
   }
   int dstate[10];
   int i=1,n,j,k,flag=0,m,q,r;
   dstate[i++]=1;
   n=i;

   dfa[1][1]=nfa[1][1];
   dfa[1][2]=nfa[1][2];
```

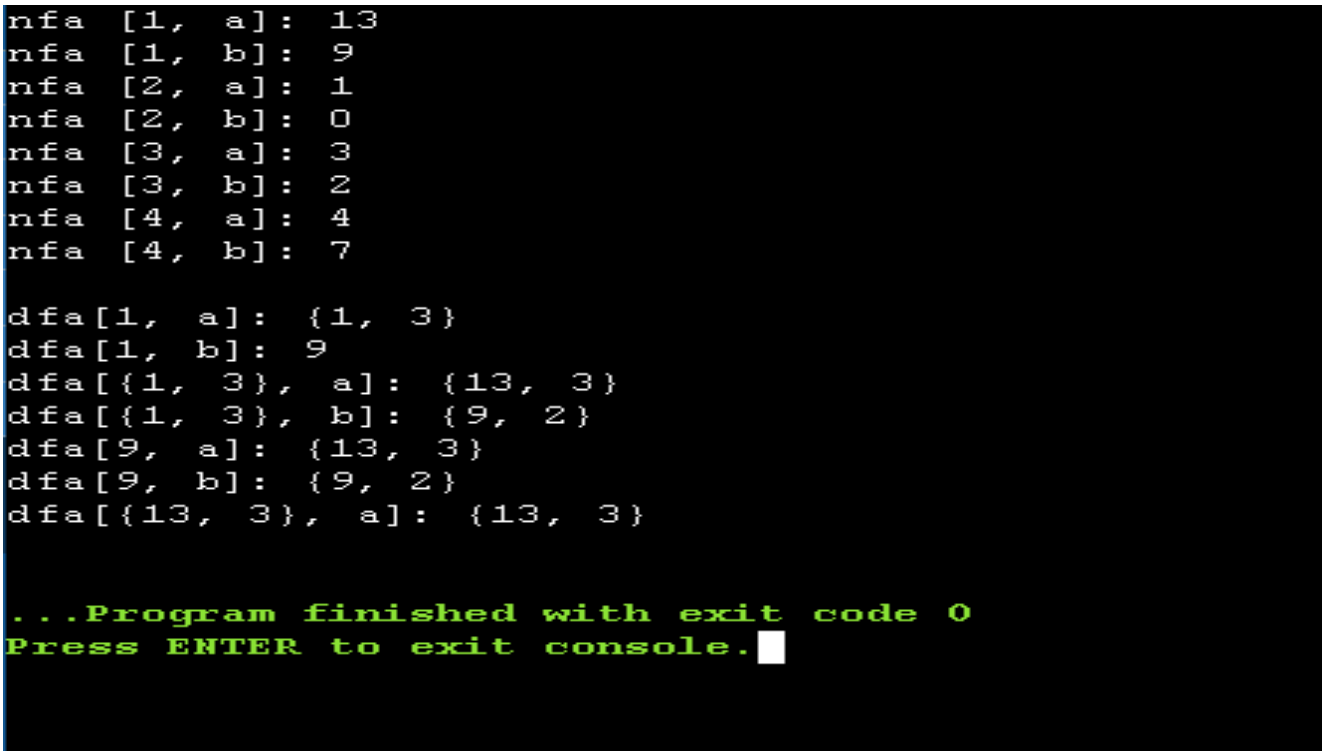
```
    cout<<"\n"<<"dfa["<<dstate[1]<<", a]: {"<<dfa[1][1]/10<<",
"<<dfa[1][1]%10<<"}";
    cout<<"\n"<<"dfa["<<dstate[1]<<", b]: "<<dfa[1][2];


  for(j=1;j<n;j++)
    {
      if(dfa[1][1]!=dstate[j])
        flag++;
    }
    if(flag==n-1)
    {
      dstate[i++]=dfa[1][1];
      n++;
    }
    flag=0;
    for(j=1;j<n;j++)
    {
       if(dfa[1][2]!=dstate[j])
          flag++;
    }
    if(flag==n-1)
    {
       dstate[i++]=dfa[1][2];
       n++;
    }
    k=2;
    while(dstate[k]!=0)
    {
      m=dstate[k];
```

```
if(m>10)
{
   q=m/10;
   r=m%10;
}
if(nfa[r][1]!=0)
    dfa[k][1]=nfa[q][1]*10+nfa[r][1];
else
   dfa[k][1]=nfa[q][1];
if(nfa[r][2]!=0)
   dfa[k][2]=nfa[q][2]*10+nfa[r][2];
else
   dfa[k][2]=nfa[q][2];


if (dstate[k] > 10){
   if (dfa[k][1] > 10){
      cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, a]:
{"<<dfa[k][1]/10 << ", " << dfa[k][1]%10 << "}";
   }
   else{
      cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, a]:
"<<dfa[k][1];
   }
}
else{
   if (dfa[k][1] > 10){
      cout<<"\n"<<"dfa["<<dstate[k] << ", a]: {"<<dfa[k][1]/10 << ", " <<
dfa[k][1]%10 << "}";
   }
   else{
```

```
            cout<<"\n"<<"dfa["<<dstate[k] << ", a]: "<<dfa[k][1];
        }
      }
      if (dstate[k] > 10){
        if (dfa[k][2] > 10){
            cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, b]:
{"<<dfa[k][2]/10 << ", " << dfa[k][2]%10 << "}";
        }
        else{
            cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, b]:
"<<dfa[k][2];
        }
      }
      else{
        if (dfa[k][1] > 10){
            cout<<"\n"<<"dfa["<<dstate[k] << ", b]: {"<<dfa[k][2]/10 << ", " <<
dfa[k][2]%10 << "}";
        }
        else{
            cout<<"\n"<<"dfa["<<dstate[k] << ", b]: "<<dfa[k][2];
        }
      }
     flag=0;
      for(j=1;j<n;j++)
      {
       if(dfa[k][1]!=dstate[j])
        flag++;
      }
    if(flag==n-1)
    {
```

```
        dstate[i++]=dfa[k][1];
        n++;
     }
    flag=0;
    for(j=1;j<n;j++)
    {
        if(dfa[k][2]!=dstate[j])
           flag++;
    }
    if(flag==n-1)
    {
        dstate[i++]=dfa[k][2];
        n++;
    }
    k++;
    }
    return 0;
}
```

Output:-




## Result:-

Thus the conversion of NFA to DFA executed successfully.

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 4
Date: 19-02-2022

# Left Factoring

**Aim:-** To implement a code for left factoring

**Procedure:-**

1) Start
2) Ask the user to enter the set of productions
3) Check for common symbols in the given set of productions by comparing with:

   $A \rightarrow aB1 | aB2$

4) If found, replace the particular productions with:

   $A \rightarrow aA'$

   $A' \rightarrow B1 | B2 | \varepsilon$

5) Display the output
6) Exit

Program:-

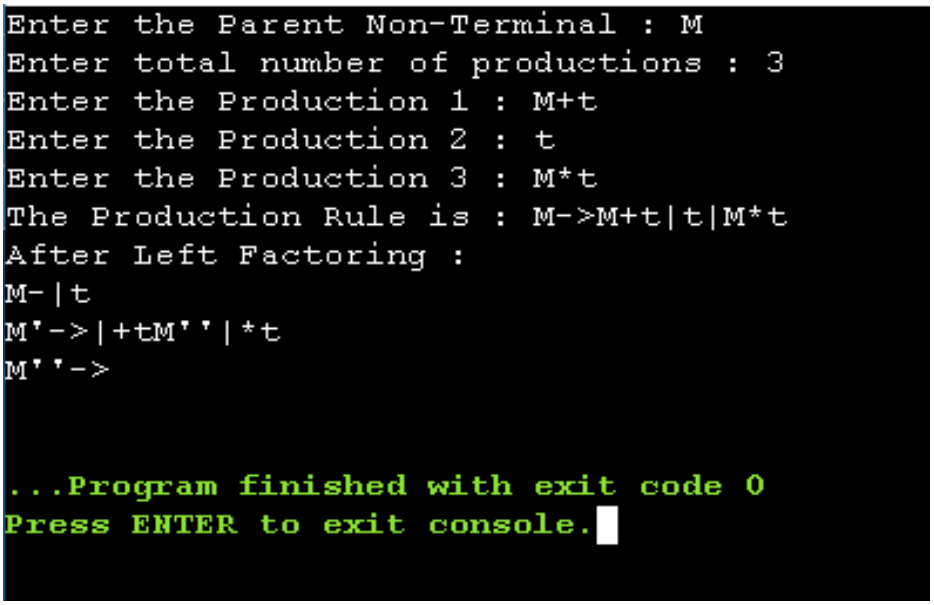
```
#include <iostream>
#include <string>
using namespace std;
int main()
{
      int n,j,l,i,m;
      int len[10] = {};
   string a, b1, b2, flag;
   char c;
   cout << "Enter the Parent Non-Terminal : ";
   cin >> c;
   a.push_back(c);
   b1 += a + "\'->";
   b2 += a + "\'\'->";;
   a += "->";
   cout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




Result:-

Thus the elimination of left factoring executed successfully.

# Left Recursion

**Aim:** To implement a code for left recursion.

**Procedure:**

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

   $A \rightarrow A\alpha_1 | A\alpha_2 | \ldots\ldots | A\alpha_m$

   $A \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n$

   Then replace it by

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

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

   $A' \rightarrow \varepsilon$

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

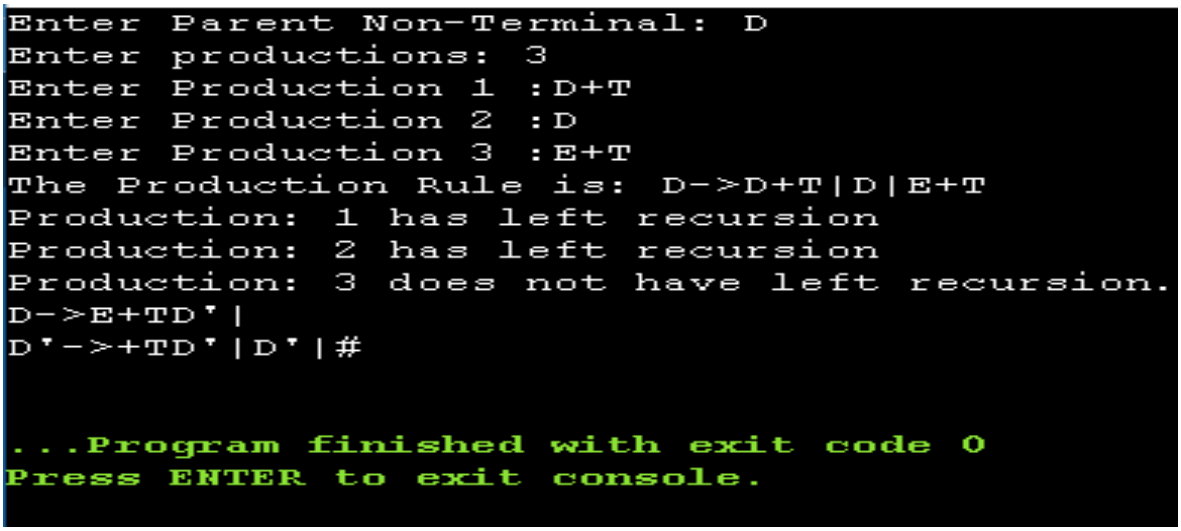
## Program:-

```
#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;
}
```

Output:-



## Result:-

Thus the elimination of left recursion executed successfully.

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 5
Date: 24-02-2022

# Computation of First and Follow

**Aim:** To write a program related to first and follow

**Procedure:-**

For computing the first:

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

Example: F -> I / id

2) We can write it as first(f) -> {(, id)

3) If X is a non-terminal like E -> T then to get first substitute T with other productions until u get a terminal as the first symbol

4) If X -> ε then add ε to first(X)

For computing the follow:

1) Always check the right side of the productions for a non-terminal, whose follow set is being found.

2) a) If that non-terminal (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.

Calculations:-

| Production | First | Follow |
|---|---|---|
| E ⇒ TR | { (, i, } | { $, ), } |
| R → +TR / # | { +, #, } | { $, ), } |
| T → FY | { (, i, } | { +, $, ), } |
| Y → *FY / # | { *, #, } | { +, $, ), } |
| F → (E) / i | { (, i, } | { *, +, $, ), } |

## Program:-

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
void followfirst(char, int, int);
void follow(char c);
void findfirst(char, int, int);
int count, n = 0;
char calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;

int main(int argc, char **argv)
{
        int jm = 0;
        int km = 0;
        int i, choice;
        char c, ch;
        count = 8;
        strcpy(production[0], "E=TR");
        strcpy(production[1], "R=+TR");
        strcpy(production[2], "R=#");
        strcpy(production[3], "T=FY");
        strcpy(production[4], "Y=*FY");
        strcpy(production[5], "Y=#");
        strcpy(production[6], "F=(E)");
        strcpy(production[7], "F=i");
```

```
int kay;
char done[count];
int ptr = -1;
for(k = 0; k < count; k++) {
        for(kay = 0; kay < 100; kay++) {
                calc_first[k][kay] = '!';
        }
}
int point1 = 0, point2, xxx;

for(k = 0; k < count; k++)
{
        c = production[k][0];
        point2 = 0;
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
                if(c == done[kay])
                        xxx = 1;
        if (xxx == 1)
                continue;
        findfirst(c, 0, 0);
        ptr += 1;
        done[ptr] = c;
        printf("\n First(%c) = { ", c);
        calc_first[point1][point2++] = c;
        for(i = 0 + jm; i < n; i++) {
                int lark = 0, chk = 0;

                for(lark = 0; lark < point2; lark++) {

                        if (first[i] == calc_first[point1][lark])
                        {
```

```
                                chk = 1;
                                break;
                        }
                }
                if(chk == 0)
                {
                        printf("%c, ", first[i]);
                        calc_first[point1][point2++] = first[i];
                }
        }
        printf("}\n");
        jm = n;
        point1++;
}
printf("\n");
printf("----------------------------------------------\n\n");
char donee[count];
ptr = -1;
for(k = 0; k < count; k++) {
        for(kay = 0; kay < 100; kay++) {
                calc_follow[k][kay] = '!';
        }
}
point1 = 0;
int land = 0;
for(e = 0; e < count; e++)
{
        ck = production[e][0];
        point2 = 0;
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
                if(ck == donee[kay])
                        xxx = 1;
```

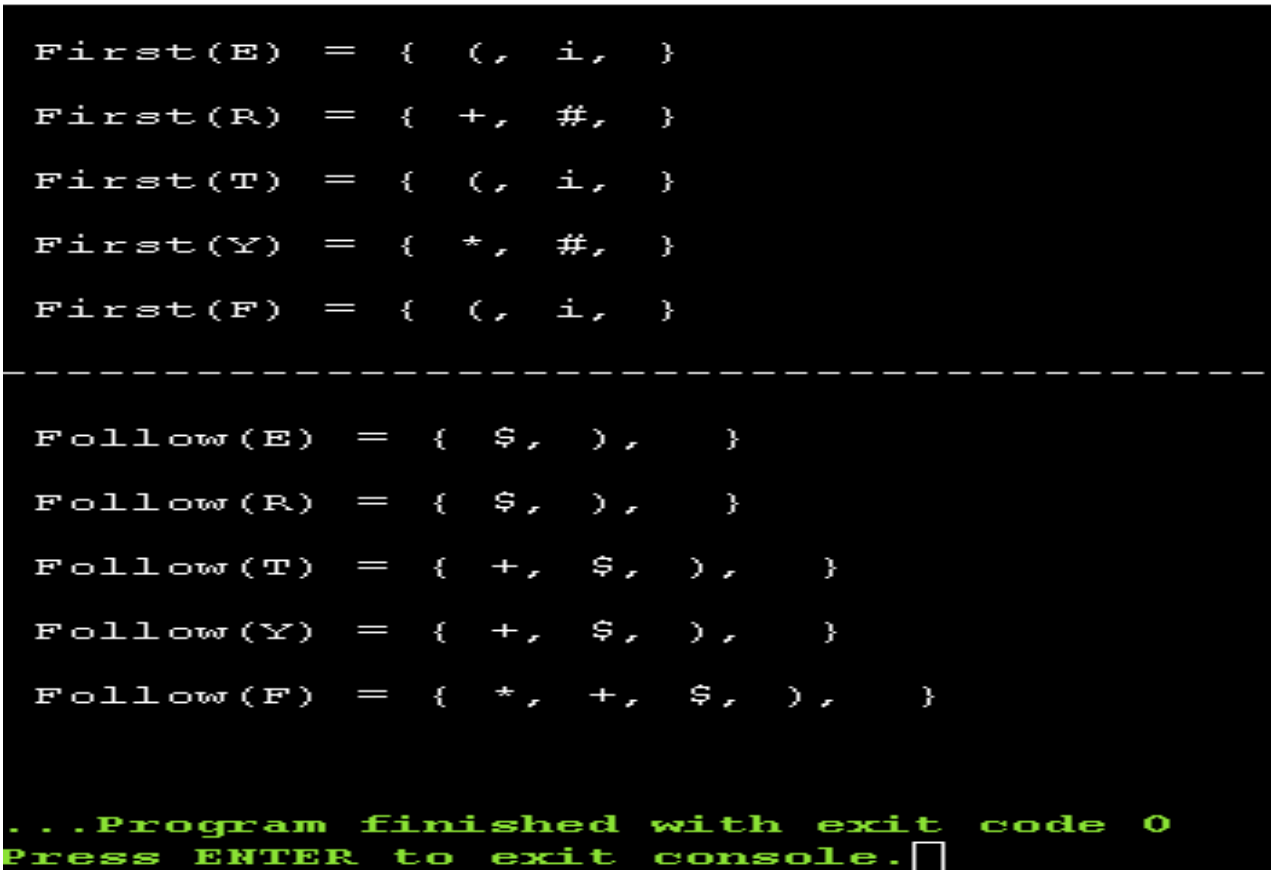
```
                if (xxx == 1)
                        continue;
                land += 1;
                follow(ck);
                ptr += 1;
                donee[ptr] = ck;
                printf(" Follow(%c) = { ", ck);
                calc_follow[point1][point2++] = ck;
                for(i = 0 + km; i < m; i++) {
                        int lark = 0, chk = 0;
                        for(lark = 0; lark < point2; lark++)
                        {
                                if (f[i] == calc_follow[point1][lark])
                                {
                                        chk = 1;
                                        break;
                                }
                        }
                        if(chk == 0)
                        {
                                printf("%c, ", f[i]);
                                calc_follow[point1][point2++] = f[i];
                        }
                }
                printf(" }\n\n");
                km = m;
                point1++;
        }
}

void follow(char c)
{
```

```
	int i, j;
	if(production[0][0] == c) {
		f[m++] = '$';
	}
	for(i = 0; i < 10; i++)
	{
		for(j = 2;j < 10; j++)
		{
			if(production[i][j] == c)
			{
				if(production[i][j+1] != '\0')
				{
					followfirst(production[i][j+1], i, (j+2));
				}

				if(production[i][j+1]=='\0' && c!=production[i][0])
				{
					follow(production[i][0]);
				}
			}
		}
	}
}

void findfirst(char c, int q1, int q2)
{
	int j;
	if(!(isupper(c))) {
		first[n++] = c;
	}
	for(j = 0; j < count; j++)
	{
		if(production[j][0] == c)
```

```
                {
                        if(production[j][2] == '#')
                        {
                                if(production[q1][q2] == '\0')
                                        first[n++] = '#';
                                else if(production[q1][q2] != '\0'
                                                && (q1 != 0 || q2 != 0))
                                {
                                        findfirst(production[q1][q2], q1, (q2+1));
                                }
                                else
                                        first[n++] = '#';
                        }
                        else if(!isupper(production[j][2]))
                        {
                                first[n++] = production[j][2];
                        }
                        else
                        {
                                findfirst(production[j][2], j, 3);
                        }
                }
        }
}

void followfirst(char c, int c1, int c2)
{
        int k;
        if(!(isupper(c)))
                f[m++] = c;
        else
        {
                int i = 0, j = 1;
```

```
                for(i = 0; i < count; i++)
                {
                        if(calc_first[i][0] == c)
                                break;
                }
                while(calc_first[i][j] != '!')
                {
                        if(calc_first[i][j] != '#')
                        {
                                f[m++] = calc_first[i][j];
                        }
                        else
                        {
                                if(production[c1][c2] == '\0')
                                {
                                        follow(production[c1][0]);
                                }
                                else
                                {
                                        followfirst(production[c1][c2], c1, c2+1);
                                }
                        }
                        j++;
                }
        }
}
```

Output:-




Result:

The first and follow sets of the non -terminals of a grammer were found successfully.

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 6
Date: 06-03-2022

# Predictive parsing table

**Aim :-** To write a program related to 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 the grammar if it is accepted or not.

**Calculations:-**

| | | |
|---|---|---|
| $E \to TE'$ | {id, c} | {$, >} |
| $E' \to +TE' \mid \epsilon$ | {+, ε} | {$, >} |
| $T \to FT'$ | {id, c} | {id} |
| $T' \to *FT' \mid \epsilon$ | {*, ε} | {+, $, >} |
| $F \to id \mid (E)$ | {id, c} | {*, +, $, >} |

| | id | + | * | ( | ) | $ |
|---|---|---|---|---|---|---|
| E | E→TE' | | | E→TE' | | |
| E' | | E'→+TE' | | | E'→ϵ | E'→ϵ |
| T | T→FT' | | | T→FT' | T→ϵ | |
| T' | | T'→ϵ | T'→*FT' | | T'→ϵ | T'→ϵ |
| F | F→id | | | F→(E) | | |

Program:-

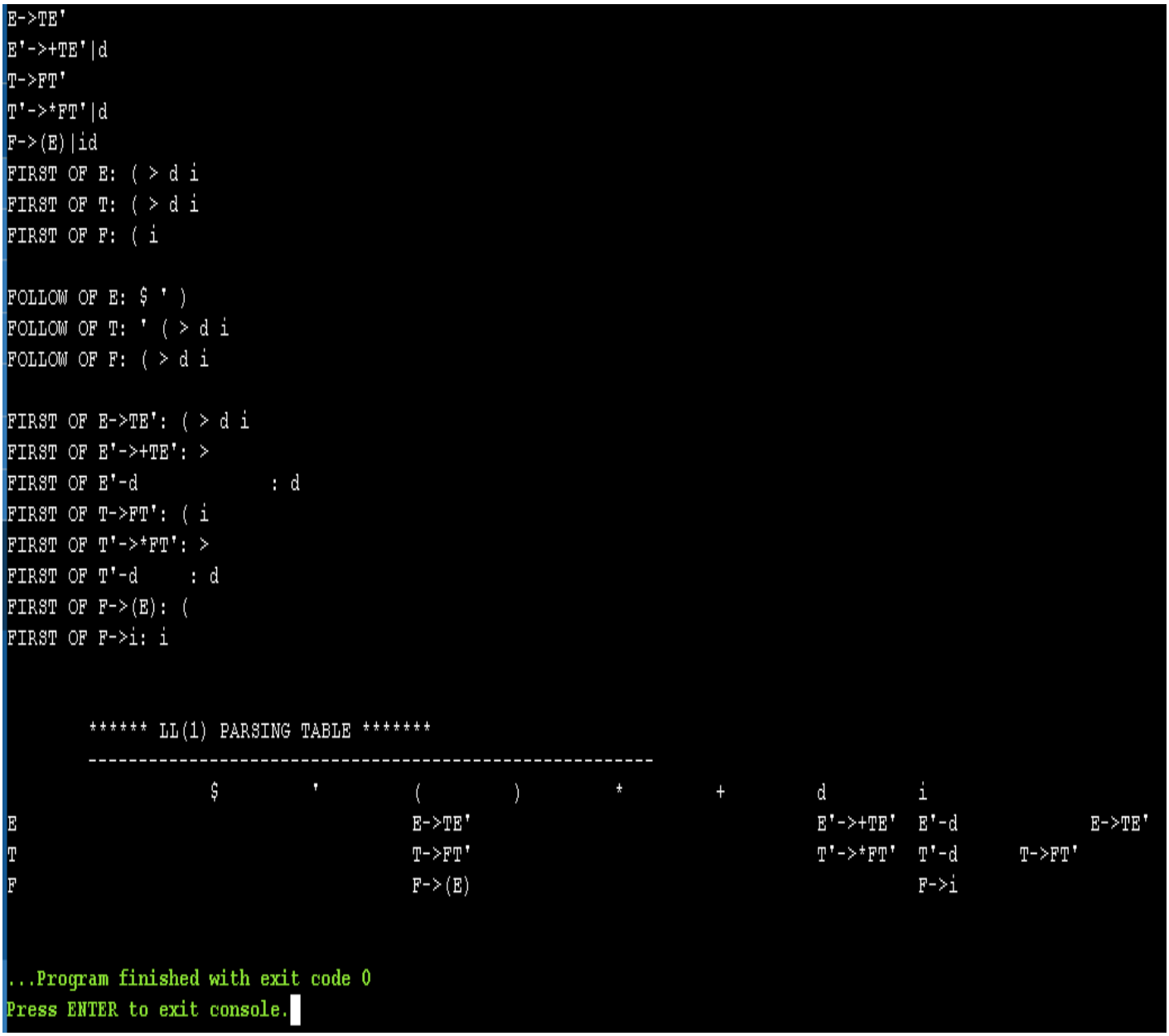
```
#include <iostream>
#include <string>
using namespace std;
int main()
{
int n,j,l,i,m;
int len[10] = {};
string a, b1, b2, flag;
char c;
cout << "Enter the Parent Non-Terminal : ";
cin >> c;
a.push_back(c);
b1 += a + "\'->";
b2 += a + "\'\'->";;
a += "->";
cout << "Enter total number of productions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




## Result:-

Thus the construction of predictive parsing table has been executed successfully.

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 7
Date: 10-03-2022

# Shift-Reduce Parser

**Aim:-** To write a program related to shift reduce parser

**Algorithm:-**

1) Start the program
2) Initialize the required variables
3) Get the number of productions from the user.
4) Perform shift operation which involves moving of symbols from input buffer on to the stack
5) Perform the reduce operation by popping out the RHS of production rule and pushing the LHS of production.
6) If the start symbol is present in the stack and the input buffer is empty then the accept the parsing action.
7) When accept action is obtained, it is means that successful parsing is done.

**Calculations:-**

Let's consider the grammar

$E \rightarrow 2E2$

$E \rightarrow 3E3$

$E \rightarrow 4$

Input string :- 32423

| Stack | Input buffer | Parsing Action |
|---|---|---|
| $ | 32423$ | Shift |
| $3 | 2423$ | Shift |
| $32 | 423$ | Shift |
| $324 | 23$ | Reduce by E→4 |
| $32E | 23$ | Shift |
| $32E2 | 3$ | Reduce by E→2E2 |
| $3E | 3$ | Shift |
| $3E3 | $ | Reduce by E→3E3 |
| $E | $ | Accept |

## Program:-

```
#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 ;
}

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->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:-

Thus the construction of Shift Reduce parser has been executed successfully.

A.Venkata Dinesh Reddy
RA1911028010098
J-1 Section

EXP NO: 5
Date: 17-03-2022

# Leading and Trailing

**Aim :-** To write a program related to Leading and Trailing.

**Procedure:-**

1) Start the program
2) For Leading, check for the first non-terminal
3) If found, print it
4) Look for the next production for the same nonterminal
5) If not found, recursively call the procedure for the Single non-terminal present before the end of production string.
6) Include it's results in the result of the non-terminal
7) For trailing, we compute same as leading but we start from the end of production from the backward.
8) Stop the program.

Calculations:-

Let the productions be:-

$E \to E+T$

$E \to T$

$T \to T*F$

$T \to F$

$F \to (E)$

$F \to id$

$\Rightarrow$

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid id$

Leading

Leading(E) = {+, Leading(T)}

= {+, *, Leading(F)}

= {+, *, (, id}

Leading(T) = {*, Leading(F)}

= {*, (, id}

Leading(F) = {(, id}

Trailing

Trailing(E) = {+, trailing(T)}

= {+, *, trailing(F)}

= {+, *, ), id}

Trailing(T) = {*, trailing(F)}

= {*, ), id}

Trailing(F) = {), id}

Result:- Successfully executed the program related to Leading and Trailing.

## Program:-

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

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

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

```
                }
            }
        }
        for(rep=0;rep<vars;rep++)
        {
            for(i=0;i<vars;i++)
            {
                for(j=0;j<gram[i].prodno;j++)
                {
                    for(m=1;m<vars;m++)
                    {
                        if(gram[i].rhs[j][0]==var[m])
                        {
                            temp=m;
                            goto out;
                        }
                    }
                    out:
                    for(k=0;k<terms;k++)
                    {
                        if(lead[temp][k]==1)
                            lead[i][k]=1;
                    }
                }
            }
        }
}
void trailing()
{
```

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

```
                        {
                                if(gram[i].rhs[j][count-1]==var[m])
                                        temp=m;
                        }
                        for(k=0;k<terms;k++)
                        {
                                if(trail[temp][k]==1)
                                        trail[i][k]=1;
                        }
                }
        }
}
void display()
{
        for(i=0;i<vars;i++)
        {
                cout<<"\nLEADING("<<gram[i].lhs<<") = ";
                for(j=0;j<terms;j++)
                {
                        if(lead[i][j]==1)
                                cout<<term[j]<<",";
                }
        }
        cout<<endl;
        for(i=0;i<vars;i++)
        {
                cout<<"\nTRAILING("<<gram[i].lhs<<") = ";
                for(j=0;j<terms;j++)
```

```
                {
                        if(trail[i][j]==1)
                                cout<<term[j]<<",";
                }
        }
}
int main()
{

        get();
        leading();
        trailing();
        display();

}
```

Output:-

```
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,

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

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 9
Date: 24/03/2022

# Compilation of LR(0) Items

**Aim:** A program to implement LR(0) items

**Procedure:-**

1) Start
2) Create structure for production with LHS and RHS
3) Open file and read input from file
4) Build state 0 from extra grammar Law $S' \rightarrow S\$$ that is all start symbol of grammar and one Dot (.) before S Symbol
5) If Dot symbol is before a non-terminal, add grammar laws that this non-terminal is in Left hand side of that Law and set Dot in before of first part of Right Hand side.
6) If state exists (a state with this Laws and same Dot position), use that instead.
7) Now find set of terminals and non-terminals in which Dot exist in before.
8) If step 7 Sel is non-empty go to 9, else go to 10
9) For each terminal / non-terminal in 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

Calculations:

E -) E+T
E -) T
T -) T*F
T -) F
F -) (E)
F -> id

Augumented grammar

E' -> E
E -) E+T
E -) T
T -) T*F
T -> F
F -) (E)
F -> id

Follow(E) = { $, +, ) }
Follow(T) = { $, +, ), * }
Follow(F) = { $, +, ), * }

Canonical LR(0) collection

$I_0$: E' -) .E
E -) .E+T
E -> .T
E -> .T*F
T -> .F
F -) .(E)
F -) .id

goto ($I_0$, E)

$I_1$: E' -) E.
E -> E.+T

goto ($I_0$, T)

$I_2$: E -> T.
T -) T.*F

goto ($I_0$, F)

$I_3$: T -> F.

goto ($I_0$, C)

$I_4$: F -) (.E)
E -) .E+T
E -) .T
T -) .F
F -) .(E)
F -) .id

goto ($I_0$, id)

$I_5$: F -) id.

goto ($I_1$, +)

$I_6$: E -) E+.T
T -) .T*F
T -) .F
F -) .(E)
F -) .id

goto ($I_2$, *)

$I_7$ : T → T * · F

F → · (E)

F → · id

goto ($I_4$, F)

$I_8$ : F → (E ·)

E → E · + T

goto ($I_6$, T)

$I_9$ : E → E + T ·

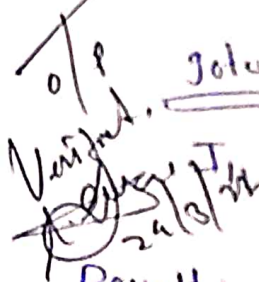
T → T · * F

goto ($I_7$, F)

$I_{10}$ : T → T * F ·

goto ($I_8$, )

$I_{11}$ : F → (E) ·



Result

The program was successfully compiled and executed.

Program:-

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

using namespace std;

char prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQR";
int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
int noitem=0;

struct Grammar
{
	char lhs;
	char rhs[8];
}g[20],item[20],clos[20][10];

int isvariable(char variable)
{
	for(int i=0;i<novar;i++)
		if(g[i].lhs==variable)
			return i+1;
	return 0;
}
void findclosure(int z, char a)
{
	int n=0,i=0,j=0,k=0,l=0;
	for(i=0;i<arr[z];i++)
```

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

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

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

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

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

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

```
{
        char list[10];
        int l=0;
        for(j=0;j<arr[z];j++)
        {
                for(k=0;k<strlen(clos[z][j].rhs)-1;k++)
                {
                        if(clos[z][j].rhs[k]=='.')
                        {
                                for(m=0;m<l;m++)
                                        if(list[m]==clos[z][j].rhs[k+1])
                                                break;
                                if(m==l)
                                        list[l++]=clos[z][j].rhs[k+1];
                        }
                }
        }
        for(int x=0;x<l;x++)
                findclosure(z,list[x]);
}
cout<<"\n THE SET OF ITEMS ARE \n\n";
for(int z=0; z<noitem; z++)
{
        cout<<"\n I"<<z<<"\n\n";
        for(j=0;j<arr[z];j++)
                cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";

}
```

}

Output:-

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


 augumented grammar

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


 I0

A->.E
E->.E+T
E->.T
T->.T*F
T->.F
F->.(E)
F->.id
```

```
  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->.id

  I5

F->i.d

  I6

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

```
F->.id

  I7

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

  I8

F->(E.)
E->E.+T

  I9

F->id.

  I10

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

  I11

T->T*F.

  I12

F->(E).


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

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 10

Date: 31-03-2022

# Intermediate code generation - postfix, prefix

**Aim:** To implement a program related to postfix, prefix

**Procedure:**

1) Start
2) Declare the set of operators
3) Initialize an empty stack
4) To convert infix to postfix, follow the following steps
5) Scan the infix expression from left to right
6) If the scanned character is an operand, output it
7) Else, if the precedence of the scanned operator is greater than the precedence of the operator in the stack, push it
8) 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
9) If the scanned character is an '(', push it to the stack
10) If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis

11) To convert Infix to Postfix follow the following steps

12) First, reverse the infix expression given in the problem

13) Scan the expression from left to right

14) Whenever the operands arrive, print them

15) If the operator arrives and the stack is found to be empty, then simply push the operator in stack.

16) Repeat steps 6 to 9 until the stack is Empty.

Calculations:

Let the expression be : (A+B)*(C+D)

=) 1) Firstly, operator precedence rules should be followed, Since parenthesis has higher precedence than the multiplication operator

2) '+' will be resolved first, and the + operator will come after AB and CD shown as below

(AB+)*(CD+)

3) Now the multiplication operator will move after CD+

4) So the postfix expression will be AB+CD+*

=) 1) In the prefix conversion the '+' operator will move before the operands AB and CD

2) So it will be (+AB)*(+CD)

3) Now multiplication operator will move before the +AB

So the prefix expression will be *+AB+CD

Result:

Successfully converted the expressions into prefix and postfix

## Infix to prefix:

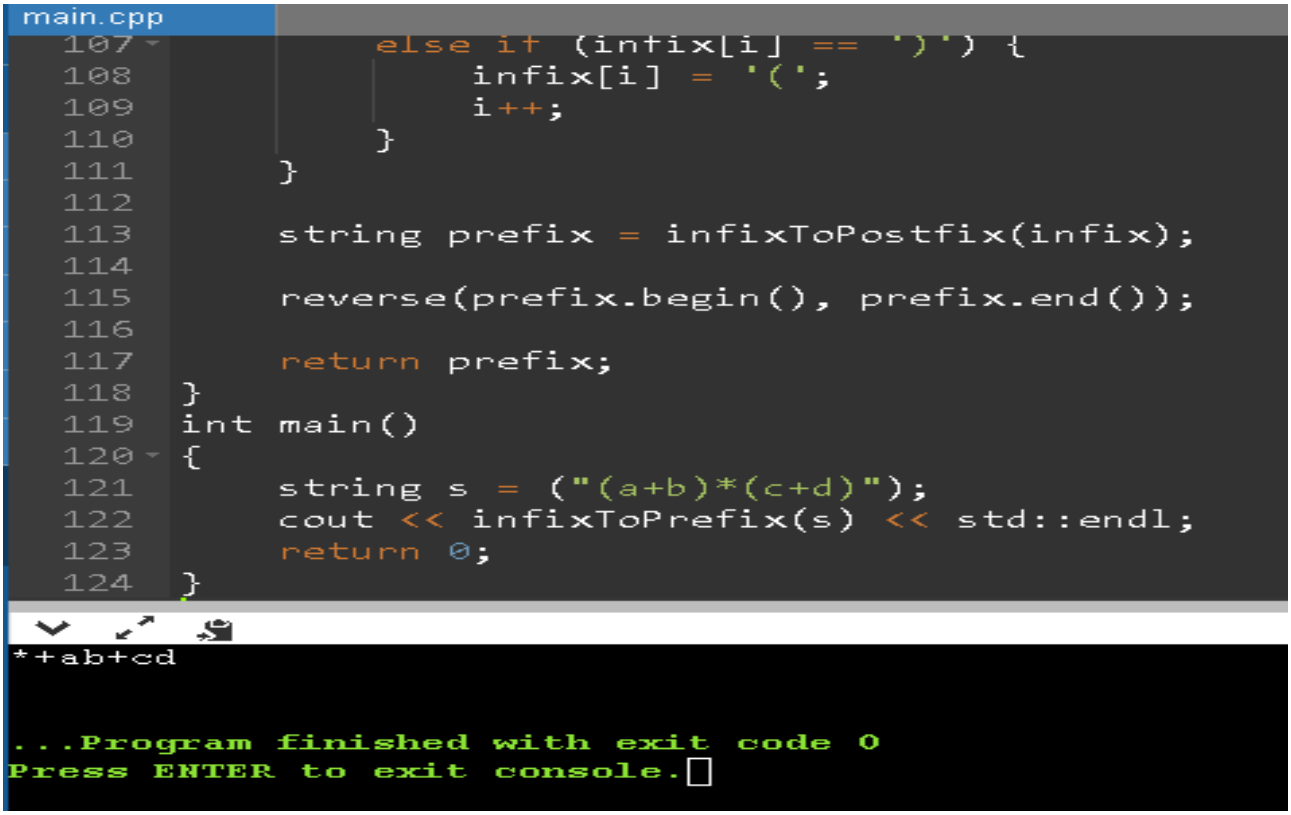
```
#include <bits/stdc++.h>
using namespace std;

bool isOperator(char c)
{
	return (!isalpha(c) && !isdigit(c));
}

int getPriority(char C)
{
	if (C == '-' || C == '+')
		return 1;
	else if (C == '*' || C == '/')
		return 2;
	else if (C == '^')
		return 3;
	return 0;
}

string infixToPostfix(string infix)
{
	infix = '(' + infix + ')';
	int l = infix.size();
	stack<char> char_stack;
	string output;

	for (int i = 0; i < l; i++) {

		// If the scanned character is an
		// operand, add it to output.
		if (isalpha(infix[i]) || isdigit(infix[i]))
```

```
        output += infix[i];

// If the scanned character is an
// ‘(‘, push it to the stack.
else if (infix[i] == '(')
        char_stack.push('(');

// If the scanned character is an
// ‘)’, pop and output from the stack
// until an ‘(‘ is encountered.
else if (infix[i] == ')') {
        while (char_stack.top() != '(') {
                output += char_stack.top();
                char_stack.pop();
        }

        // Remove '(' from the stack
        char_stack.pop();
}

// Operator found
else
{
        if (isOperator(char_stack.top()))
        {
                if(infix[i] == '^')
                {
                        while (getPriority(infix[i]) <= getPriority(char_stack.top()))
                        {
                                output += char_stack.top();
                                char_stack.pop();
                        }
```

```
                        }
                        else
                        {
                                while (getPriority(infix[i]) < getPriority(char_stack.top()))
                                {
                                        output += char_stack.top();
                                        char_stack.pop();
                                }

                        }

                        // Push current Operator on stack
                        char_stack.push(infix[i]);
                }
        }
    }
    while(!char_stack.empty()){
            output += char_stack.top();
            char_stack.pop();
    }
    return output;
}

string infixToPrefix(string infix)
{
    /* Reverse String
    * Replace ( with ) and vice versa
    * Get Postfix
    * Reverse Postfix * */
    int l = infix.size();

    // Reverse infix
    reverse(infix.begin(), infix.end());
```

```
        // Replace ( with ) and vice versa
        for (int i = 0; i < l; i++) {

                if (infix[i] == '(') {
                        infix[i] = ')';
                        i++;
                }
                else if (infix[i] == ')') {
                        infix[i] = '(';
                        i++;
                }
        }

        string prefix = infixToPostfix(infix);

        reverse(prefix.begin(), prefix.end());

        return prefix;
}
int main()
{
        string s = ("(a+b)*(c+d)");
        cout << infixToPrefix(s) << std::endl;
        return 0;
}
```

## Output:-




## Infix to postfix:

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

// Stack type
struct Stack
{
    int top;
    unsigned capacity;
    int* array;
};

// Stack Operations
struct Stack* createStack( unsigned capacity )
```

```
{
        struct Stack* stack = (struct Stack*)
                malloc(sizeof(struct Stack));

        if (!stack)
                return NULL;

        stack->top = -1;
        stack->capacity = capacity;

        stack->array = (int*) malloc(stack->capacity *
                                                        sizeof(int));

        return stack;
}
int isEmpty(struct Stack* stack)
{
        return stack->top == -1 ;
}
char peek(struct Stack* stack)
{
        return stack->array[stack->top];
}
char pop(struct Stack* stack)
{
        if (!isEmpty(stack))
                return stack->array[stack->top--] ;
        return '$';
}
void push(struct Stack* stack, char op)
{
        stack->array[++stack->top] = op;
}
```

```
// A utility function to check if
// the given character is operand
int isOperand(char ch)
{
	return (ch >= 'a' && ch <= 'z') ||
		(ch >= 'A' && ch <= 'Z');
}

// A utility function to return
// precedence of a given operator
// Higher returned value means
// higher precedence
int Prec(char ch)
{
	switch (ch)
	{
	case '+':
	case '-':
		return 1;

	case '*':
	case '/':
		return 2;

	case '^':
		return 3;
	}
	return -1;
}
```

```
// The main function that
// converts given infix expression
// to postfix expression.
int infixToPostfix(char* exp)
{
	int i, k;

	// Create a stack of capacity
	// equal to expression size
	struct Stack* stack = createStack(strlen(exp));
	if(!stack) // See if stack was created successfully
		return -1 ;

	for (i = 0, k = -1; exp[i]; ++i)
	{

		// If the scanned character is
		// an operand, add it to output.
		if (isOperand(exp[i]))
			exp[++k] = exp[i];

		// If the scanned character is an
		// ‘(‘, push it to the stack.
		else if (exp[i] == '(')
			push(stack, exp[i]);

		// If the scanned character is an ‘)’,
		// pop and output from the stack
		// until an ‘(‘ is encountered.
		else if (exp[i] == ')')
		{
			while (!isEmpty(stack) && peek(stack) != '('
				exp[++k] = pop(stack);
```

```
                if (!isEmpty(stack) && peek(stack) != '(')
                        return -1; // invalid expression
                else
                        pop(stack);
        }
        else // an operator is encountered
        {
                while (!isEmpty(stack) &&
                        Prec(exp[i]) <= Prec(peek(stack)))
                        exp[++k] = pop(stack);
                push(stack, exp[i]);
        }

    }

    while (!isEmpty(stack))
        exp[++k] = pop(stack );

    exp[++k] = '\0';
    printf( "%s", exp );
  }
int main()
{
    char exp[] = "(a+b)*(c+d)";
    infixToPostfix(exp);
    return 0;
}
```

```
ab+cd+*

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

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

Exp No: 11
Date : 31-03-2022

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

**Aim** : To implement the code related to quadruple, triple, indirect triple.

**Algorithm:**

1) Start
2) It is a sequence of three-address statements as input.
3) For each three address statements of the form a:=b op c perform the various actions. These are as follows
   i) Invoke a function getreg to find out the location L where the result of computation b op c should be stored
   ii) Consult the address description for y to determine the value y.
   iii) Generate the instruction OP z', L where z' is used to show the current location of z. If z in both, then prefer a register to a memory location
   iv) If the current value of y or z have no next uses or not live on exit from the block.

Calculations:

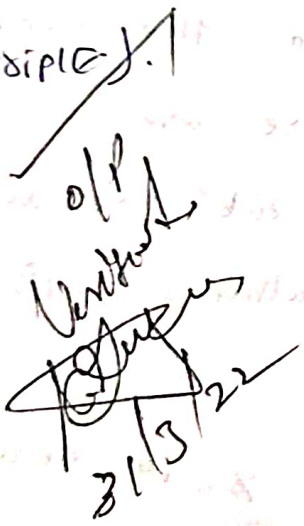
Let the expression be: a+b*c-d is:

** T1 = B * C
T2 = A + T2
T3 = T3 + D **

=) a = b+c-d

t1 = b-1 c
t2 = t1 - d
a = t2

Result:- Successfully executed the program related to the three address code (Quadruple, triple, indirect triple).



## Code:

```
#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("\tThe 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("\tt%d=%s*%s\n",c,a,b);
if(j[i]=='/')
printf("\tt%d=%s/%s\n",c,a,b);
if(j[i]=='+')
printf("\tt%d=%s+%s\n",c,a,b);if(j[i]=='-')
printf("\tt%d=%s-%s\n",c,a,b);
if(j[i]=='=')
printf("\t%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);}
```

## Output:

```c
#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("\tThe 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];
```

input

```
nter the expression:a=b+c-d
        The Intermediate code is:
        t1=b+c
        t2=t1-d
        a=t2

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

A. Venkata Dinesh Reddy
RA1911028010098
J-1 Section

EXP NO: 12
Date: 07-04-2022

# Implementation of DAG

**Aim :-** A program to implementation of DAG

**Procedure:-**

1) The leaves of a graph are labeled by a unique identifier and that identifier can be variable names or constants
2) Interior nodes of the graph are labeled by an operator symbol.
3) Nodes are also given a sequence of identifiers for labels to store the computed value.
4) If y operand is undefined then create node(y).
5) If z operand is undefined then for case(i) create node(z).
6) For case(i), create node(op) whose right child is node(z) and left child is node(y).
7) For case(ii), check whether there is node(op) with one child node(y).
8) For case (iii), node n will be node(y).
9) For Node(x) delete x from the list of identifiers.
10). Append x to attached identifiers list for the node n found in step 2. Finally set node(x) to n.

Calculations:-

Three possible scenarios for building a DAG

Case 1 :- X = Y op Z

Case 2 :- X = op Y

Case 3 : X = Y

Let the expression is a = b*-c+b*-c

Z := b*

Y = c+Z

X = c-Y

W = X-c

a = W a := $

Result: Successfully executed the program related to DAG.

## Program:-

```
#include<stdio.h>
#include<string.h>
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
 int pos;
 char op;
}k[15];
void main()
{

 printf("\t\tINTERMEDIATE CODE GENERATION OF DAG\n\n");

 scanf("%s",str);
 printf("The intermediate code:\t\tExpression\n");
 findopr();
 explore();

}
void findopr()
{
 for(i=0;str[i]!='\0';i++)
  if(str[i]==':')
```

```
 {
 k[j].pos=i;
 k[j++].op=':';
 }
for(i=0;str[i]!='\0';i++)
 if(str[i]=='/')
 {
 k[j].pos=i;
 k[j++].op='/';
 }
for(i=0;str[i]!='\0';i++)
 if(str[i]=='*')
 {
 k[j].pos=i;
 k[j++].op='*';
 }
for(i=0;str[i]!='\0';i++)
 if(str[i]=='+')
 {
 k[j].pos=i;
 k[j++].op='+';
 }
for(i=0;str[i]!='\0';i++)
 if(str[i]=='-')
 {
 k[j].pos=i;
 k[j++].op='-';
 }
}
```

```
void explore()
{
 i=1;
 while(k[i].op!='\0')
 {
  fleft(k[i].pos);
  fright(k[i].pos);
  str[k[i].pos]=tmpch--;
  printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
  for(j=0;j <strlen(str);j++)
   if(str[j]!='$')
    printf("%c",str[j]);
  printf("\n");
  i++;
 }
 fright(-1);
 if(no==0)
 {
  fleft(strlen(str));
  printf("\t%s := %s",right,left);
 }
 printf("\t%s :=  %c",right,str[k[--i].pos]);

}
void fleft(int x)
{
 int w=0,flag=0;
 x--;
 while(x!= -1 &&str[x]!= '+'
```

```
&&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&str[x]!='/'&&str[x]!=':')
   {
    if(str[x]!='$'&& flag==0)
    {
    left[w++]=str[x];
    left[w]='\0';
    str[x]='$';
    flag=1;
    }
    x--;
   }
  }
  void fright(int x)
  {
   int w=0,flag=0;
   x++;
   while(x!= -1 && str[x]!=
'+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-
'&&str[x]!='/')
   {
    if(str[x]!='$'&& flag==0)
    {
    right[w++]=str[x];
    right[w]='\0';
    str[x]='$';
    flag=1;
    }
    x++;
   }
  }
```

## Output:-

main.c

```
#include<stdio.h>
#include<string.h>
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
 int pos;
 char op;
}k[15];
void main()
{
```

input

```
                INTERMEDIATE CODE GENERATION OF DAG

a=b*-c+b*-c
The intermediate code:          Expression
        Z := b*          a=b*-c+Z-c
        Y := c+Z                 a=b*-Y-c
        X := c-Y                 a=b*X-c
        W := X-c                 a=b*W
        a := W   a :=   $

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

## Result:-

Thus the construction of DAG has been  executed successfully.

A. Venkata Dinesh Reddy
RA1911028010098
J1 section

Exp No: 13
Date: 08-04-2022

# Implement any one storage allocation strategies

**Aim:** To implement any of the stack storage allocation strategies

**Procedure:**

1) Initially check whether the stack is empty
2) Insert an element into the stack using push operation
3) Insert more elements onto the stack until stack becomes full
4) Delete an element from the stack using pop operation
5) Display the elements in the stack
6) Top the stack element will be displayed.

**Calculations:**

```
Enter your choice (1-5) : 1
Enter the element : 10
Do you want to enter more elements? (y/n) : y
Enter the element : 20
Do you want to enter more elements ? (y/n) : n
The list is created
Enter your choice(1-5) : 2
10 -> 20 -> NULL
```

**Result:** Thus, we have successfully implemented the stack storage allocation strategy using heap.

## Program:-

```
#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:-

main.c

```c
#include<stdio.h>
#include<stdlib.h>
#define TRUE 1
#define FALSE 0
typedef struct Heap
{
int data;
```

input

```
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)3
h1.insert a node as a head noden2.insert a node as a head noden3.insert a node at intermediate position in t6he list
enter your choice for insertion of node:3

enter the elements which you want to insert1

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)
```