Exp.No: 1 Regular Expression to NFA

AIM:

To write a program for converting Regular Expression to NFA using C++ Language.

ALGORITHM:

1. Start

2. Get the input from the user

3. Initialize separate variables and functions for Postfix , Display and NFA

4. Create separate methods for different operators like +,*, .

5. By using Switch case Initialize different cases for the input

6. For ' . ' operator Initialize a separate method by using various stack functions do the same for the other operators like ' * ' and ' + '.

7. Regular expression is in the form like a.b (or) a+b

8. Display the output

9. Stop

PROGRAM:

```
#include<iostream>

#include<conio.h>

#include<stdio.h>
#include<string.h>
char reg[20];
void postfix();
void e_nfa();
void disp(int,char,int);
using namespace std;
int main()
{
//clrscr();
cin>>reg;
postfix();
cout<<reg<<endl;
e_nfa();
getch();
return 0;
}
void postfix()
{
char string[10],stack[10];
int string_n=0,stack_n=0;
int n=0;
strcat(reg,"X");
while(reg[n]!='\0')
{switch(reg[n])
{
case 'a' : string [string_n]='a';
string_n++;
string[string_n]='\0';
break;
case 'b' : string[string_n]='b';
string_n++;
string[string_n]='\0';
break;
case '*' : string[string_n]='*';
string_n++;
string[string_n]='\0';
break;
case '(' : stack[stack_n]='(';
stack_n++;
break;
case ')' : stack_n--;
```

```
while(stack[stack_n]!='(')
{
string[string_n]=stack[stack_n];
stack[stack_n]='\0';
string_n++;
string[string_n]='\0';
stack_n--;
}
stack[stack_n]='\0';
break;
case 'X' : while(stack_n!=0)
{
stack_n--;
string[string_n]=stack[stack_n];
stack[stack_n]='\0';
string_n++;
string[string_n]='\0';
}
break;
case '+' : if(stack[stack_n-1]!='+'&&stack[stack_n-
1]!='.')
{
stack[stack_n]='+';
stack_n++;
stack[stack_n]='\0';
break;
}
else
{
string[string_n]=stack[stack_n-1];
string_n++;
stack[stack_n-1]='+';
break;
}
case '.' : if(stack[stack_n-1]!='+'&&stack[stack_n-
1]!='.')
{
stack[stack_n]='.';
stack_n++;
stack[stack_n]='\0';
break;
}
else
{
string[string_n]=stack[stack_n-1];
string_n++;
stack[stack_n-1]='.';
break;
```

```
}

default:break;
}

n++;
}
strcpy(reg,string);
}
void e_nfa()
{

int strt[3],last[3],s,l;
int n=0,x=0,i=-1;
while(reg[n]!='\0')
{
switch(reg[n])
{

case 'a':i++;
strt[i]=x++;
last[i]=x++;
disp(strt[i],'a',last[i]);
break;

case 'b':i++;
strt[i]=x++;
last[i]=x++;
disp(strt[i],'b',last[i]);
break;

case '+' : s=x++;
l=x++;
disp(s,'e',strt[i]);
disp(s,'e',strt[i-1]);
disp(last[i],'e',l);
disp(last[i-1],'e',l);
i--;
strt[i]=s;
last[i]=l;
break;

case '.' : disp(last[i-1],'e',strt[i]);
last[i-1]=last[i];
i--;
break;

case '*' : s=x++;
l=x++;
```

```
disp(s,'e',strt[i]);

disp(s,'e',l);
disp(last[i],'e',strt[i]);
disp(last[i],'e',l);
strt[i]=s;
last[i]=l;
break;


default:break;
}
n++;
}
cout<<i<<" "<<strt[i]<<" "<<last[i];
}
void disp(int qs,char a,int qf)
{
cout<<qs<<"-->\t"<<a<<"-->\t"<<qf<<"\n";
}
```

RESULT:

The C++ program to convert Regular expression to NFA has been successfully executed.

Exp.No: 2 NFA to DFA CONVERSION

AIM:

To write a program for converting NFA to DFA using C++ Language.

ALGORITHM:

1. Start

2. Get the input from the user

3. Implement the following sudo code:

Set the only state in SDFA to “unmarked”

while SDFA contains an unmarked state do

Let T be that unmarked state

for each a in % do

S = #-Closure(MoveNFA(T,a))

if S is not in SDFA already then

Add S to SDFA (as an “unmarked” state)

endIf

Set MoveDFA(T,a) to S

endFor

endWhile

for each S in SDFA do

if any s&S is a final state in the NFA then

Mark S an a final state in the DFA

endIf

endFor

4. Print the result and Stop the program.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#define STATES  256
#define SYMBOLS 20
int N_symbols;
int NFA_states;
char *NFAtab[STATES][SYMBOLS];int DFA_states; /* number of DFA states */
int DFAtab[STATES][SYMBOLS];
 void put_dfa_table(
    int tab[][SYMBOLS], /* DFA table */
    int nstates,    /* number of states */
    int nsymbols)   /* number of input symbols */
{

    int i, j;
    puts("STATE TRANSITION TABLE");
    printf("     | ");
    for (i = 0; i < nsymbols; i++) printf("  %c  ", '0'+i);
    printf("\n-----+--");
    for (i = 0; i < nsymbols; i++) printf("-----");
    printf("\n");
    for (i = 0; i < nstates; i++) {
        printf("  %c  | ", 'A'+i);  /* state */
        for (j = 0; j < nsymbols; j++)
            printf("  %c  ", 'A'+tab[i][j]);
        printf("\n");
    }
}

void init_NFA_table()
{
    NFAtab[0][0] = "12";
    NFAtab[0][1] = "13";
    NFAtab[1][0] = "12";
    NFAtab[1][1] = "13";
    NFAtab[2][0] = "4";
    NFAtab[2][1] = "";
    NFAtab[3][0] = "";
    NFAtab[3][1] = "4";
    NFAtab[4][0] = "4";
    NFAtab[4][1] = "4";
    NFA_states = 5;
    DFA_states = 0;
    N_symbols = 2;
}
```

```
void string_merge(char *s, char *t)
{
    char temp[STATES], *r=temp, *p=s;
    while (*p && *t) {
        if (*p == *t) {
            *r++ = *p++; t++;
        } else if (*p < *t) {
            *r++ = *p++;
        } else
            *r++ = *t++;
    }
    *r = '\0';
    if (*p) strcat(r, p);
    else if (*t) strcat(r, t);
    strcpy(s, temp);
}
void get_next_state(char *nextstates, char *cur_states,
    char *nfa[STATES][SYMBOLS], int n_nfa, int symbol)
{

    int i;
    char temp[STATES];

    temp[0] = '\0';
    for (i = 0; i < strlen(cur_states); i++)
        string_merge(temp, nfa[cur_states[i]-'0'][symbol]);
    strcpy(nextstates, temp);

}
int state_index(char *state, char statename[][STATES], int *pn)
{

    int i;

    if (!*state) return -1; /* no next state */

    for (i = 0; i < *pn; i++)
        if (!strcmp(state, statename[i])) return i;

    strcpy(statename[i], state);    /* new state-name */
    return (*pn)++;

}
```

```
int nfa_to_dfa(char *nfa[STATES][SYMBOLS], int n_nfa,
    int n_sym, int dfa[][SYMBOLS])
{

    char statename[STATES][STATES];
    int i = 0;  /* current index of DFA */
    int n = 1;  /* number of DFA states */
    char nextstate[STATES];
    int j;
    strcpy(statename[0], "0");  /* start state */
    for (i = 0; i < n; i++) {    /* for each DFA state */
        for (j = 0; j < n_sym; j++) {    /* for each input symbol */
            get_next_state(nextstate, statename[i], nfa, n_nfa, j);
            dfa[i][j] = state_index(nextstate, statename, &n);
        }
    }
    return n;   /* number of DFA states */
}
int main()
{

    init_NFA_table();
    DFA_states = nfa_to_dfa(NFAtab, NFA_states, N_symbols, DFAtab);
    put_dfa_table(DFAtab, DFA_states, N_symbols);
    return 0;

}
```

RESULT:

The C++ program to convert NFA to DFA has been successfully executed.

Exp.No: 3 FIRST AND FOLLOW

AIM:

To write a program to perform first and follow using C language.

ALGORITHM:

For computing the first:

If X is a terminal then FIRST(X) = {X}

Example: F -> (E) | id

We can write it as FIRST(F) -> { (, id }

2. If X is a non terminal like E -> T then to get FIRST(E) substitute T with other productions until you get a terminal as the first symbol

3. 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. (never see the left side).

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

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

PROGRAM:

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
int n,m=0,p,i=0,j=0;
char a[10][10],f[10];
void follow(char c);
void first(char c);
int main(){
int i,z;
char c,ch;
//clrscr();
printf("Enter the no of prooductions:\n");
scanf("%d",&n);
printf("Enter the productions:\n");
for(i=0;i<n;i++)
scanf("%s%c",a[i],&ch);
do{
m=0;
printf("Enter the elemets whose fisrt & follow is to be found:");
scanf("%c",&c);
first(c);
printf("First(%c)={",c);
for(i=0;i<m;i++)
printf("%c",f[i]);
printf("}\n");
strcpy(f," ");
//flushall();
m=0;
follow(c);
printf("Follow(%c)={",c);
for(i=0;i<m;i++)
printf("%c",f[i]);
printf("}\n");
printf("Continue(0/1)?");
scanf("%d%c",&z,&ch);
}while(z==1);
return(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]=='$')
follow(a[k][0]);
else if(islower(a[k][2]))
f[m++]=a[k][2];
else first(a[k][2]);
}
}
}
void follow(char c)
{
if(a[0][0]==c)
f[m++]='$';
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]);
}
}
}
}
```

RESULT:

The C program to perform first and follow has been successfully executed.

Exp.No: 4 ELIMINATION OF LEFT RECURSION

AIM:

To write a program in C to eliminate left recursion.

ALGORITHM:

1. Start the program.

2. Initialize the arrays for taking input from the user.

3. Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.

4. Prompt the user to input the right production for non-terminals.

5. Eliminate left recursion using the following rules:-

A->Aα1| Aα2 | . . . . . |Aαm

A->β1| β2| . . . . .| βn

Then replace it by:

A'-> βi A' i=1,2,3,.....m

A'-> αj A' j=1,2,3,.....n

A'-> Ɛ

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

7. Stop.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
int main()
{
char s[100];
printf("Enter production to remove left recursion\n");
scanf("%s",s);
int l=strlen(s),i,j=0,k;
char a[100],b[100],A;
A=s[0];
for(i=4;i<=l;i++)
{
if(s[i]=='|')
{
break;
}
else
{
a[j]=s[i];
j++;
}
}
i++;
j=0;
for(k=i;k<l;k++)
{
b[j]=s[k];
j++;
}
printf("ELEMENTS AFTER LEFT RECURSION  (NOTE:EMPTY STATE IS
REPRESENTED AS #)\n");
printf("%c->%s%c'\n",A,b,A);
printf("%c'->%s%c'|#",A,a,A);
return 0;
}
```

RESULT:

The C program to eliminate left recursion has been successfully executed.

Exp.No: 5 ELIMINATION OF LEFT FACTORING

AIM:

To write a C++ program to remove left factoring from a set of given productions.

ALGORITHM:

1. Start

2. Ask the user to enter the set of productions from which the left factoring is to be removed.

3. Check for left factoring in the given set of productions by comparing with:

A->aB1|aB2

4. If found, replace the particular productions with:

A->aA’

A’->B1 | B2|ε

5. Display the output

6. Exit

PROGRAM:

```
#include<stdio.h>
 #include<string.h>
 int main()
 {
     char
gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
     int i,j=0,k=0,l=0,pos;
     printf("Enter Production : A->");
     gets(gram);
     for(i=0;gram[i]!='|';i++,j++)
         part1[j]=gram[i];
     part1[j]='\0';
     for(j=++i,i=0;gram[j]!='\0';j++,i++)
        part2[i]=gram[j];
     part2[i]='\0';
    for(i=0;i<strlen(part1)||i<strlen(part2);i++)
    {
         if(part1[i]==part2[i])
         {
             modifiedGram[k]=part1[i];
             k++;
             pos=i+1;
         }
     }
     for(i=pos,j=0;part1[i]!='\0';i++,j++){
         newGram[j]=part1[i];
     }
     newGram[j++]='|';
     for(i=pos;part2[i]!='\0';i++,j++){
         newGram[j]=part2[i];
     }
     modifiedGram[k]='X';
     modifiedGram[++k]='\0';
     newGram[j]='\0';
     printf("\n A->%s",modifiedGram);
     printf("\n X->%s\n",newGram);
 }
```

RESULT:

The C++ program to remove left factoring from a set of given productions has been successfully executed.

Exp.No: 6 LEXICAL ANALYZER

AIM:

To write a c program that works as a lexical analyzer.

ALGORITHM:

1. Start the program.

2. Take input superated by spaces.

3. Whenever you identify any of '+ % - * /' display it as operator.

4. If the word is a keyword display it as a keyword.

5. Else display "Identifier".

6.Stop.

PROGRAM:

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

using namespace std;

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[] = "+-*/%=";
ifstream fin("program.txt");
int i,j=0;
if(!fin.is_open())
{
cout<<"error while opening the file\n";
exit(0);
}

while(!fin.eof()){
    ch = fin.get();

for(i = 0; i < 6; ++i)
{
    if(ch == operators[i])
    cout<<ch<<" is operator\n";
 }
```

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



    if(isKeyword(buffer) == 1)
    cout<<buffer<<" is keyword\n";
    else
    cout<<buffer<<" is indentifier\n";
    }

}
fin.close();
return 0;
}
```

RESULT:

A c-program which works as a lexical analyzer is successfully implemented.

Exp.No: 7 LEADING AND TRAILING

AIM:

To write a C++ program to compute the leading and trailing of the given grammar.

ALGORITHM:

1. For Leading, check for the first non-terminal.

2. If found, print it.

3. Look for next production for the same non-terminal.

4. If not found, recursively call the procedure for the single non-terminal present before the comma or End Of Production String.

5. Include it's results in the result of this non-terminal.

6. For trailing, we compute same as leading but we start from the end of the production to the beginning.

7. Stop

PROGRAM:

```
#include <iostream>
#include <cstring>
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:";
  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";
  }
  return 0;
}
```

RESULT:

The C++ program to compute the leading and trailing of the given grammar has been successfully executed.

Exp.No: 8 LR(0) ITEM CONSTRUCTION

AIM:

To perform LR(0) Item construction on a production using C programming.

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<string.h>

#include<stdio.h>

int axn[][6][2]={
                    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
                    {{-1,-1},{100,6},{-1,-1},{-1,-1},{-1,-1},{102,102}},
                    {{-1,-1},{101,2},{100,7},{-1,-1},{101,2},{101,2}},
                    {{-1,-1},{101,4},{101,4},{-1,-1},{101,4},{101,4}},
                    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
                    {{100,5},{101,6},{101,6},{-1,-1},{101,6},{101,6}},
                    {{100,5},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1}},
                    {{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
                    {{-1,-1},{100,6},{-1,-1},{-1,-1},{100,11},{-1,-1}},
                    {{-1,-1},{101,1},{100,7},{-1,-1},{101,1},{101,1}},
                    {{-1,-1},{101,3},{101,3},{-1,-1},{101,3},{101,3}},
                    {{-1,-1},{101,5},{101,5},{-1,-1},{101,5},{101,5}}
};

int gotot[12][3]={1,2,3,-1,-1,-1,-1,-1,-1,-1,-1,-1,8,2,3,-1,-1,-1,-1,
                    9,3,-1,-1,10,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

int a[10];
char b[10];
int top=-1,btop=-1,i;
void push(int k)
{
        if(top<9)
        a[++top]=k;
}
void pushb(char k)
{
        if(btop<9)
        b[++btop]=k;
}
char TOS()
{
        return a[top];
}
void pop()
{
        if(top>=0)
        top--;
}
```

```
void popb()
{
	if(btop>=0)
	b[btop--]='\0';
}
void display()
{
	for(i=0;i<=top;i++)
		printf("%d%c",a[i],b[i]);
}
void display1(char p[],int m)
{
	int l;
	printf("\t\t");
	for(l=m;p[l]!='\0';l++)
		printf("%c",p[l]);
		printf("\n");
}
void error(){
	printf("\n\nSyntax Error");
}
void reduce(int p){
	int len,k,ad;
	char src,*dest;
	switch(p)
	{
		case 1:dest="E+T";
			src='E';
			break;
		case 2:dest="T";
			src='E';
			break;
		case 3:dest="T*F";
			src='T';
			break;
		case 4:dest="F";
			src='T';
			break;
		case 5:dest="(E)";
			src='F';
			break;
		case 6:dest="i";
			src='F';
			break;
		default:dest="\0";
			src='\0';
			break;
	}
```

```
		for(k=0;k<strlen(dest);k++)
		{
			pop();
			popb();
		}
		pushb(src);
		switch(src)
		{
			case 'E': ad=0;
				break;
			case 'T': ad=1;
				break;
			case 'F': ad=2;
				break;
			default: ad=-1;
				break;
		}
		push(gotot[TOS()][ad]);
}
int main()
{
		int j,st,ic;
		char ip[20]="\0",an;

		printf("Enter any String :- ");
		gets(ip);
		push(0);
		display();
		printf("\t%s\n",ip);
		for(j=0;ip[j]!='\0';)
		{
			st=TOS();
			an=ip[j];
			if(an>='a'&an<='z')
				ic=0;
			else if(an=='+')
				ic=1;
			else if(an=='*')
				ic=2;
			else if(an=='(')
				ic=3;
			else if(an==')')
				ic=4;
			else if(an=='$')
				ic=5;
			else
			{
				error();
```

```
                    break;
                }
                if(axn[st][ic][0]==100)
                {
                    pushb(an);
                    push(axn[st][ic][1]);
                    display();
                    j++;
                    display1(ip,j);
                }
                if(axn[st][ic][0]==101)
                {
                    reduce(axn[st][ic][1]);
                    display();
                    display1(ip,j);
                }
                if(axn[st][ic][1]==102)
                {
                    printf("Given String is Accepted");
                    break;
                }
        }

        return 0;
}
```

RESULT:

The c- program to compute the LR(0) of the given grammar is successfully executed.

Exp.No: 9 SHIFT REDUCE PARSING

AIM:

To write a C program to perform Shift Reduce Parsing.

ALGORITHM:

1. Start the program.

2. Initialize the required variables.

3. Enter the input symbol.

4. Perform the following:

for top-of-stack symbol, s, and next input symbol, a shift x: (x is a STATE number) push a, then x on the top of the stack and advance ip to point to the next input symbol.

reduce y: (y is a PRODUCTION number) Assume that the production is of the form A ==> beta

pop 2 * |beta| symbols of the stack. At this point the top of the stack should be a state number, say s'. push A, then goto of T[s',A] (a state number) on the top of the stack. Output the production A ==> beta.

5. Print if string is accepted or not.

6. Stop the program.

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];
// Rules can be E->2E2 , E->3E3 , E->4
void check()
{
   strcpy(ac,"REDUCE TO E -> ");
   for(z = 0; z < c; z++)
   {
      //checking for producing rule E->4
      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++)
   {
      //checking for another production
      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++)
   {
      //checking for E->3E3
      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");
}
```

RESULT:

The C program to implement shift reduce parsing has been successfully executed.

Exp.No: 10 PREDICTIVE PARSING TABLE

AIM:

To write a C program to construct a predictive parsing table.

ALGORITHM:

1. Start the program.

2. Initialize the required variables.

3. Get the number of coordinates and productions from the user.

4. Perform the following

for (each production A → α in G) {

for (each terminal a in FIRST(α))

add A → α to M[A, a];

if (ε is in FIRST(α))

for (each symbol b in FOLLOW(A))

add A → α to M[A, b];

5. Print the resulting stack.

6. Print if the grammar is accepted or not.

7. Exit the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char fin[10][20],st[10][20],ft[20][20],fol[20][20];
int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;

printf("Enter the no. of Productions :\n");
scanf("%d",&n);
printf("Enter the productions in a grammar\n");
for(i=0;i<n;i++)
scanf("%s",st[i]);
for(i=0;i<n;i++)
fol[i][0]='\0';
for(s=0;s<n;s++)
{
for(i=0;i<n;i++)
{
```

```
j=3;
l=0;
a=0;
l1:if(!((st[i][j]>64)&&(st[i][j]<91)))
{
for(m=0;m<l;m++)
{
if(ft[i][m]==st[i][j])
goto s1;
}
ft[i][l]=st[i][j];
l=l+1;
s1:j=j+1;
}
else
{
if(s>0)
{
while(st[i][j]!=st[a][0])
{
a++;
```

```
}
b=0;
while(ft[a][b]!='\0')
{
for(m=0;m<l;m++)
{
if(ft[i][m]==ft[a][b])
goto s2;
}
ft[i][l]=ft[a][b];
l=l+1;
s2:b=b+1;
}
}
}
while(st[i][j]!='\0')
{
if(st[i][j]=='|')
{
j=j+1;
goto l1;
```

```
}
j=j+1;
}
ft[i][l]='\0';
}
}
printf("FIRST :\n");
for(i=0;i<n;i++)
printf("FIRST[%c]={%s}\n",st[i][0],ft[i]);
fol[0][0]='$';
for(i=0;i<n;i++)
{
k=0;
j=3;
if(i==0)
l=1;
else
l=0;
k1:while((st[i][0]!=st[k][j])&&(k<n))
{
if(st[k][j]=='\0')
```

```
{
k++;
j=2;
}
j++;
}
j=j+1;
if(st[i][0]==st[k][j-1])
{
if((st[k][j]!='|')&&(st[k][j]!='\0'))
{
a=0;
if(!((st[k][j]>64)&&(st[k][j]<91)))
{
for(m=0;m<l;m++)
{
if(fol[i][m]==st[k][j])
goto q3;
}
int o=0;
fol[i][l]=st[k][j];
```

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

```
l=l+1;
}
else
e=1;
q2:p++;
}
if(e==1)
{
e=0;
goto a1;
}
}
}
else
{
a1:c=0;
a=0;
while(st[k][0]!=st[a][0])
{
a++;
```

```
}
while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))
{
for(m=0;m<l;m++)
{
if(fol[i][m]==fol[a][c])
goto q1;
}
fol[i][l]=fol[a][c];
l++;
q1:c++;
}
}
goto k1;
}
fol[i][l]='\0';
}
printf("FOLLOW :\n");
for(i=0;i<n;i++)
printf("FOLLOW[%c]={%s}\n",st[i][0],fol[i]);
printf("\n");
```

```
s=0;
for(i=0;i<n;i++)
{
j=3;
while(st[i][j]!='\0')
{
if((st[i][j-1]=='|')||(j==3))
{
for(p=0;p<=2;p++)
{
fin[s][p]=st[i][p];
}
t=j;
for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));p++)
{
fin[s][p]=st[i][j];
j++;
}
fin[s][p]='\0';
if(st[i][k]=='@')
{
```

```
b=0;
a=0;
while(st[a][0]!=st[i][0])
{
a++;
}
while(fol[a][b]!='\0')
{
printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);
b++;
}
}
else if(!((st[i][t]>64)&&(st[i][t]<91)))
printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);
else
{
b=0;
a=0;
while(st[a][0]!=st[i][3])
{
a++;
```

```
}
while(ft[a][b]!='\0')
{
printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);
b++;
}
}
s++;
}
if(st[i][j]=='|')
j++;
}
}
getch();
}
```

RESULT:

The C program to construct predictive parsing table has been successfully executed