COMPILER DESIGN LAB (RCS-652)

BY:SHREYA SRIVASTAVA

UNIV ROLL NO: 1816110198

BRANCH & YEAR: CSE 3RD YEAR

FACULTY TEACHER: MR. PRASHANT NARESH

Program 1

AIM: WAP to check whether the entered string is accepted or not for a given grammar.

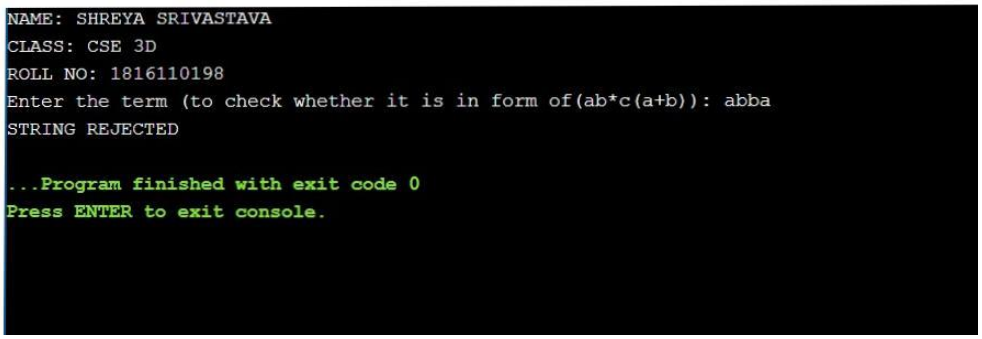
PROGRAM:

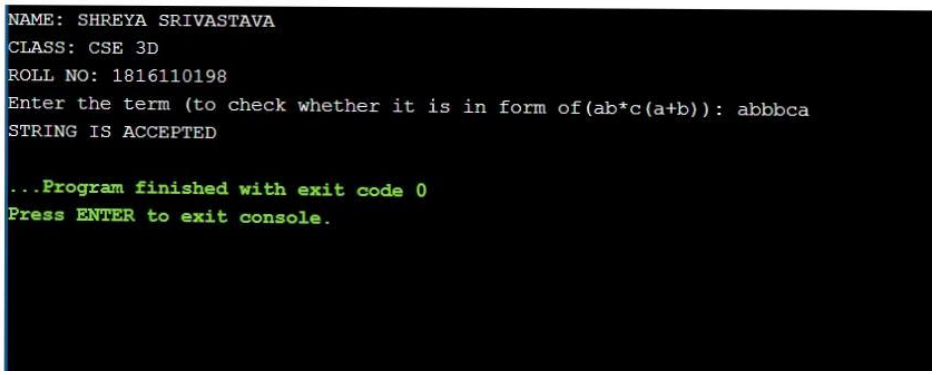
Strings acceptable by grammar are of form: ab*c(a+b)

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{int i,n;
char a[20];
printf("NAME: SHREYA SRIVASTAVA");
printf("\nCLASS: CSE 3D");
printf("\nROLL NO: 1816110198");
printf("\nEnter the term (to check whether it is in form of(ab*c(a+b)): ");
scanf("%s",&a);
n=strlen(a);
if(a[0]=='a'&& (a[n-1]=='a'|| a[n-1]=='b') && a[n-2]=='c')
{
for(i=1;i<n-2;i++)
{if(a[i]=='b')
continue;
else
{printf("STRING IS REJECTED");
```

```
exit(0);
}
}
printf("STRING IS ACCEPTED");
}
else
printf("STRING REJECTED");
getch();
}
```

OUTPUT:





Program 2

AIM: WAP to convert infix expression to postfix expression.

Expression: A+(C*D)*F

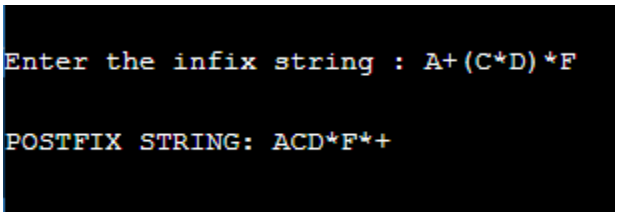
PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
char stack[20];
int top = -1;
void push(char x)
{
    stack[++top] = x;
}
char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}
```

```
int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
}
void main()
{
    clrscr();
    char exp[20];
    char *e, x;
    printf("\nEnter the infix string : ");
    scanf("%s",exp);
    e = exp;
    printf("\nPOSTFIX STRING: ");
    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '('
                printf("%c", x);
```

```
            }
            else
            {
                while(priority(stack[top]) >= priority(*e))
                    printf("%c",pop());
                push(*e);
            }
            e++;
        }
        while(top != -1)
        {
            printf("%c",pop());
        }
        printf("\n");
    }
```

OUTPUT:



Program 3

AIM: WAP to convert infix expression to prefix expression.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char str1[]="A+(C*D)*F";
char str[]="F*(D*C)+A";
char stack[10];
int top=-1;
void push(char s)
{
  top=top+1;
  stack[top]=s;
}
char pop()
{
  char item;
   item=stack[top];
   top--;
   return(item);
}
int precede(char c)
{
  if(c==47)
   return(5);
   if(c==42)
   return(4);
```

```
    if(c==43)
    return(3);
     else
     return(2);
}
void main()
{
  char prefix[10];
  int l, i=0, j=0;
  char s, temp;
  printf("INFIX STRING: ");
  puts(str);
  l=strlen(str);
  push('#');
  while(i<l)
  {
    s=str[i];
    switch(s)
     {
          case '(':
           push(s);
           break;
          case ')':
           temp=pop();
           while(temp!='(')
           {
            prefix[j]=temp;
```

```
            j++;
            temp=pop();
           }
           break;
         case '+':
         case '-':
         case '*':
         case '/':
          while(precede(stack[top])>=precede(s))
          {
                temp=pop();
                prefix[j]=temp;
                j++;
           }
         push(s);
         break;
         default:
         prefix[j++]=s;
         break;
    }
 i++;
}
while(top>0)
  {
        temp=pop();
        prefix[j++]=temp;
   }
```

```
  prefix[j++]='\0';
  printf("\nPREFIX STRING: ");
  for(i=6;i>=0;i--)
  printf("%c", prefix[i]);
  getch();
}
```

OUTPUT:

```
INFIX STRING: F*(D*C)+A
PREFIX STRING: +A**CDF
```

Program 4

AIM: WAP to find the no. of tokens and list them according to their category in an expression (given/entered)

Eg: a= b+c*23-56^2

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
int con=0, var=0, op=0;
void check(char c)
{
 if(isalpha(c))
 var++;
```

```
  if(c==47||c==42||c==43||c==45||c==61||c==94)
  op++;
}
void main()
{
  clrscr();
  char str[13];
  char c;
  printf("\nENTER STRING: ");
  scanf("%s", &str);
  for(int i=0; i<13; i++)
  {
   c=str[i];
   check(c);
  }
  for(int i=0; i<13; i++)
  {
   if(isdigit(str[i])&&isdigit(str[i+1]))
    {
     i=i+2;
     con++;
    }
    else if(isdigit(str[i]))
    con++;
  }
  printf("\n Operators: %d \nVariables: %d \nConstants: %d" , op, var, con);
  printf("\nTotal tokens=%d", op+var+con);
```

```
  getch();
}
```

OUTPUT:

```
ENTER STRING: a= b+c*23-56^2

 Operators: 1
Variables: 1
Constants: 0
Total tokens=2
```

Program 5

AIM: WAP to construct an NFA from a regular expression (given) and display the transition table of NFA constructed.

(1) What is FSM.
(2) What is transition diagram.
(3) What is E transition.
(4) What is Thomsson rule.

Given regular expression: (a/b)*

PROGRAM:

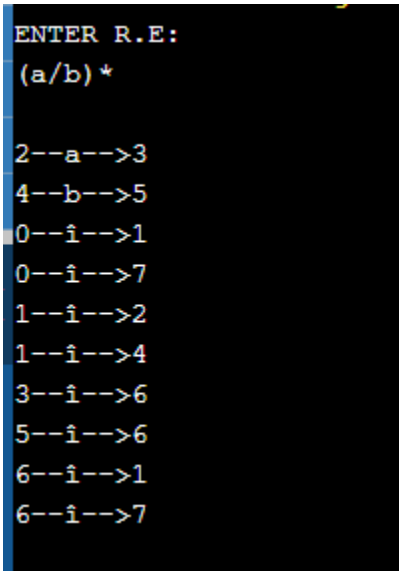
```
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<string.h>
void main()
{
```

```
clrscr();
char s[10];
int n,init=0,fin=1;
cout<<"ENTER R.E : \n”;
gets(s);
n=strlen(s);
for(int i=0;i<n;i++)
{
  if(s[i]=='*')
  fin+=2;
  if(s[i]=='.')
  fin+=1;
  if(s[i]=='/')
  fin+=4;
 }
char c=238;
i=0;
int ch;
if(s[0]>=97&&s[0]<=122)
ch=1;
if(s[0]=='('&&s[4]==')')
ch=2;
switch(ch)
{
 case 1:
 if(s[i+1]=='/')
  {
```

```
   if(s[i+2]>=97 && s[i+2]<=122)
    {
      cout<<"\n"<<init+2<<"--"<<s[i]<<"-->"<<init+3;
      cout<<"\n"<<init+4<<"--"<<s[i+2]<<"-->"<<init+5;
      goto pt1;
    }
   }
  case 2:
  if(s[i+1]>=97 && s[i+1]<=122)
  if(s[i+2]=='/')
 {
   if(s[i+3]>=97 && s[i+3]<=122)
   {
     cout<<"\n"<<init+2<<"--"<<s[i+1]<<"-->"<<init+3;
     cout<<"\n"<<init+4<<"--"<<s[i+3]<<"-->"<<init+5;
     if(s[i+5]=='*')
      {
        goto pt;
      }
    else
    goto pt1;
    }
 }
}
pt:
cout<<"\n"<<init<<"--"<<c<<"-->"<<init+1;
cout<<"\n"<<init<<"--"<<c<<"-->"<<fin;
```

```
pt1:
cout<<"\n"<<init+1<<"--"<<c<<"-->"<<init+2;
cout<<"\n"<<init+1<<"--"<<c<<"-->"<<init+4;
cout<<"\n"<<init+3<<"--"<<c<<"-->"<<init+6;
cout<<"\n"<<init+5<<"--"<<c<<"-->"<<init+6;
cout<<"\n"<<init+6<<"--"<<c<<"-->"<<init+1;
cout<<"\n"<<init+6<<"--"<<c<<"-->"<<fin;
getch();
}
```

OUTPUT:



Program 6

AIM: WAP to compute LEADING and TRAILING sets of a grammar(given).

Grammar: E→ E+T | T

T→ T*F | F

F→ (E) | id

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:";
    cin >> n;
    cout << "Enter the productions\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;
```

}

OUTPUT :

```
Enter the no of productions:6
Enter the productions
E->E+E
E->T
T->T*F
F->(E)
T->F
F->i
Leading[E]      {+,*,(,i,}
Leading[T]      {*,(,i,}
Leading[F]      {(,i,}
Trailing[E]     {+,*,),i,}
Trailing[T]     {*,),i,}
Trailing[F]     {),i,}
```

Program 7

AIM: WAP to calculate FIRST and FOLLOW.

PROGRAM:

FIRST:

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

int count,n=0;
char prodn[10][10],first[10];
```

```
void First(char ch)
{int j;
   /*if(!isupper(ch))
   {
      first[n++]=ch;
   }*/
   for(j=0;j<count;j++)
   {
      if(prodn[j][0]==ch)
      {if(prodn[j][2]=='$')
      {
         first[n++]='$';
      }
      else if(islower(prodn[j][2]))
      {
         first[n++]=prodn[j][2];
      }
      else
      First(prodn[j][2]);
      }
   }
}

void main()
{int i,choice;
char c,ch;
```

```
printf("Enter the no of productions: ");
scanf("%d",&count);
printf("\nEnter %d the production, epsilon=$ : ",count);
for(i=0;i<count;i++)
scanf("%s%c",prodn[i],&ch);
do{
   n=0;
   printf("Element :");
   scanf("%c",&c);
   First(c);
   printf("\nFIRST(%c)={",c);
   for(i=0;i<n;i++)
   {printf("%c,",first[i]);
   }
   printf("}\n");
   printf("Press 1 to continue");
   scanf("%d%c",&choice,&ch);
}while(choice==1);
}
```

OUTPUT:

```
Enter the no of productions: 3

Enter 3 the production, epsilon=$ : A=ab
B=bC
C=c
Element :A

FIRST(A)={a,}
Press 1 to continue1
Element :B

FIRST(B)={b,}
Press 1 to continue1
Element :C

FIRST(C)={c,}
Press 1 to continue0
```

FOLLOW:

```
#include<stdio.h>
#include<string.h>
int n,m=0,p,i=0,j=0;
char prodn[10][10],follow[10];
void Follow(char c)
{

  if(prodn[0][0]==c)
  follow[m++]='$';
  for(i=0;i<n;i++)
  {
    for(j=2;j<strlen(prodn[i]);j++)
    {
      if(prodn[i][j]==c)
```

```
        {
            if(prodn[i][j+1]!='\0')
            first(prodn[i][j+1]);

            if(prodn[i][j+1]=='\0'&&c!=prodn[i][0])
            Follow(prodn[i][0]);

        }
      }
   }
}
void first(char c)
{
     int k;
            if(!(isupper(c)))
            follow[m++]=c;
            for(k=0;k<n;k++)
            {
            if(prodn[k][0]==c)
            {
            if(prodn[k][2]=='$')
            Follow(prodn[i][0]);
            else if(islower(prodn[k][2]))
            follow[m++]=prodn[k][2];
            else first(prodn[k][2]);
            }
            }

}
```

```
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",prodn[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 ",follow[i]);
  printf(" }\n");
  printf("Do you want to continue?(press 1 to continue)");
  scanf("%d%c",&z,&ch);
 }
 while(z==1);
}
```

OUTPUT:

```
Enter the no.of productions:2
Enter the productions(epsilon=$):
A=Ab
B=Ca
Enter the element whose FOLLOW is to be found:A
FOLLOW(A) = { $ b  }
Do you want to continue?(press 1 to continue)1
Enter the element whose FOLLOW is to be found:C
FOLLOW(C) = { a  }
Do you want to continue?(press 1 to continue)0
```

Program 8

AIM: WAP in C to check whether the Grammar is Left-recursive and remove left recursion.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#define SIZE 10
int main()
{
   char non_terminal;
   char beta, alpha;
   int num;
   char production[10][SIZE];
   int index = 3; /* starting of the string following "->" */
   printf("Enter Number of Production : ");
   scanf("%d", &num);
   printf("Enter the grammar as E->E-A :\n");
   for (int i = 0; i < num; i++)
```

```
  {
    scanf("%s", production[i]);
  }
  for (int i = 0; i < num; i++)
  {
    printf("\nGRAMMAR : : : %s", production[i]);
    non_terminal = production[i][0];
    if (non_terminal == production[i][index])
    {
      alpha = production[i][index + 1];
      printf(" is left recursive.\n");
      while (production[i][index] != 0 && production[i][index] != '|')
        index++;
      if (production[i][index] != 0)
      {
        beta = production[i][index + 1];
        printf("Grammar without left recursion:\n");
        printf("%c->%c%c\'", non_terminal, beta, non_terminal);
        printf("\n%c\'->%c%c\'|E\n", non_terminal, alpha, non_terminal);
      }
      else
        printf(" can't be reduced\n");
    }
    else
      printf(" is not left recursive.\n");
    index = 3;
  }
}
```

OUTPUT:

```
Enter Number of Production : 4
Enter the grammar as E->E-A :
E->EA|A
A->AT|a
T->a
E->i

GRAMMAR : : : E->EA|A is left recursive.
Grammar without left recursion:
E->AE'
E'->AE'|E

GRAMMAR : : : A->AT|a is left recursive.
Grammar without left recursion:
A->aA'
A'->TA'|E

GRAMMAR : : : T->a is not left recursive.

GRAMMAR : : : E->i is not left recursive.
```

Prgramm-9

AIM: WAP in C to draw a SLR parsing table for a given grammar

PROGRAM:

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

```
#include <stdio.h>

using namespace std;

char terminals[100] = {};
int no_t;
char non_terminals[100] = {};
int no_nt;
char goto_table[100][100];
char reduce[20][20];
char follow[20][20];
char fo_co[20][20];
char first[20][20];

struct state
{
   int prod_count;
   char prod[100][100] = {{}};
};

void add_dots(struct state *I)
{
   for (int i = 0; i < I->prod_count; i++)
   {
      for (int j = 99; j > 3; j--)
         I->prod[i][j] = I->prod[i][j - 1];
      I->prod[i][3] = '.';
```

```
    }
}

void augument(struct state *S, struct state *I)
{
    if (I->prod[0][0] == 'S')
        strcpy(S->prod[0], "Z->.S");
    else
    {
        strcpy(S->prod[0], "S->.");
        S->prod[0][4] = I->prod[0][0];
    }
    S->prod_count++;
}

void get_prods(struct state *I)
{
    cout << "Enter the number of productions:\n";
    cin >> I->prod_count;
    cout << "Enter the number of non terminals:" << endl;
    cin >> no_nt;
    cout << "Enter the non terminals one by one:" << endl;
    for (int i = 0; i < no_nt; i++)
        cin >> non_terminals[i];
    cout << "Enter the number of terminals:" << endl;
    cin >> no_t;
    cout << "Enter the terminals (single lettered) one by one:" << endl;
```

```
    for (int i = 0; i < no_t; i++)
        cin >> terminals[i];
    cout << "Enter the productions one by one in form (S->ABc):\n";
    for (int i = 0; i < I->prod_count; i++)
    {
        cin >> I->prod[i];
    }
}

bool is_non_terminal(char a)
{
    if (a >= 'A' && a <= 'Z')
        return true;
    else
        return false;
}

bool in_state(struct state *I, char *a)
{
    for (int i = 0; i < I->prod_count; i++)
    {
        if (!strcmp(I->prod[i], a))
            return true;
    }
    return false;
}
```

```
char char_after_dot(char a[100])
{
    char b;
    for (int i = 0; i < strlen(a); i++)
        if (a[i] == '.')
        {
            b = a[i + 1];
            return b;
        }
}

char *move_dot(char b[100], int len)
{
    char a[100] = {};
    strcpy(a, b);
    for (int i = 0; i < len; i++)
    {
        if (a[i] == '.')
        {
            swap(a[i], a[i + 1]);
            break;
        }
    }
    return &a[0];
}

bool same_state(struct state *I0, struct state *I)
```

```
{

    if (Io->prod_count != I->prod_count)
        return false;

    for (int i = 0; i < Io->prod_count; i++)
    {
        int flag = 0;
        for (int j = 0; j < I->prod_count; j++)
            if (strcmp(Io->prod[i], I->prod[j]) == 0)
                flag = 1;
        if (flag == 0)
            return false;
    }
    return true;
}

void closure(struct state *I, struct state *Io)
{
    char a = {};
    for (int i = 0; i < Io->prod_count; i++)
    {
        a = char_after_dot(Io->prod[i]);
        if (is_non_terminal(a))
        {
            for (int j = 0; j < I->prod_count; j++)
            {
```

```
            if (I->prod[j][0] == a)
            {
               if (!in_state(I0, I->prod[j]))
               {
                  strcpy(I0->prod[I0->prod_count], I->prod[j]);
                  I0->prod_count++;
               }
            }
         }
      }
   }
}

void goto_state(struct state *I, struct state *S, char a)
{
   int time = 1;
   for (int i = 0; i < I->prod_count; i++)
   {
      if (char_after_dot(I->prod[i]) == a)
      {
         if (time == 1)
         {
            time++;
         }
         strcpy(S->prod[S->prod_count], move_dot(I->prod[i], strlen(I-
>prod[i])));
         S->prod_count++;
```

```
        }
    }
}

void print_prods(struct state *I)
{
    for (int i = 0; i < I->prod_count; i++)
        printf("%s\n", I->prod[i]);
    cout << endl;
}

bool in_array(char a[20], char b)
{
    for (int i = 0; i < strlen(a); i++)
        if (a[i] == b)
            return true;
    return false;
}

char *chars_after_dots(struct state *I)
{
    char a[20] = {};
    for (int i = 0; i < I->prod_count; i++)
    {
        if (!in_array(a, char_after_dot(I->prod[i])))
        {
            a[strlen(a)] = char_after_dot(I->prod[i]);
```

```
        }
    }
    return &a[0];
}

void cleanup_prods(struct state *I)
{
    char a[100] = {};
    for (int i = 0; i < I->prod_count; i++)
        strcpy(I->prod[i], a);
    I->prod_count = 0;
}

int return_index(char a)
{
    for (int i = 0; i < no_t; i++)
        if (terminals[i] == a)
            return i;
    for (int i = 0; i < no_nt; i++)
        if (non_terminals[i] == a)
            return no_t + i;
}

void print_shift_table(int state_count)
{
    cout << endl
         << "********Shift Actions*********" << endl
```

```
    << endl;
cout << "\t";
for (int i = 0; i < no_t; i++)
   cout << terminals[i] << "\t";
for (int i = 0; i < no_nt; i++)
   cout << non_terminals[i] << "\t";
cout << endl;
for (int i = 0; i < state_count; i++)
{
   int arr[no_nt + no_t] = {-1};
   for (int j = 0; j < state_count; j++)
   {
      if (goto_table[i][j] != '~')
      {
         arr[return_index(goto_table[i][j])] = j;
      }
   }
   cout << "I" << i << "\t";
   for (int j = 0; j < no_nt + no_t; j++)
   {
      if (i == 1 && j == no_t - 1)
         cout << "ACC"
              << "\t";
      if (arr[j] == -1 || arr[j] == 0)
         cout << "\t";
      else
      {
```

```
            if (j < no_t)
               cout << "S" << arr[j] << "\t";
            else
               cout << arr[j] << "\t";
         }
      }
      cout << "\n";
   }
}

int get_index(char c, char *a)
{
   for (int i = 0; i < strlen(a); i++)
      if (a[i] == c)
         return i;
}

void add_dot_at_end(struct state *I)
{
   for (int i = 0; i < I->prod_count; i++)
   {
      strcat(I->prod[i], ".");
   }
}

void add_to_first(int n, char b)
{
```

```
    for (int i = 0; i < strlen(first[n]); i++)
        if (first[n][i] == b)
            return;
    first[n][strlen(first[n])] = b;
}

void add_to_first(int m, int n)
{
    for (int i = 0; i < strlen(first[n]); i++)
    {
        int flag = 0;
        for (int j = 0; j < strlen(first[m]); j++)
        {
            if (first[n][i] == first[m][j])
                flag = 1;
        }
        if (flag == 0)
            add_to_first(m, first[n][i]);
    }
}

void add_to_follow(int n, char b)
{
    for (int i = 0; i < strlen(follow[n]); i++)
        if (follow[n][i] == b)
            return;
    follow[n][strlen(follow[n])] = b;
```

```
}

void add_to_follow(int m, int n)
{
   for (int i = 0; i < strlen(follow[n]); i++)
   {
      int flag = 0;
      for (int j = 0; j < strlen(follow[m]); j++)
      {
         if (follow[n][i] == follow[m][j])
            flag = 1;
      }
      if (flag == 0)
         add_to_follow(m, follow[n][i]);
   }
}

void add_to_follow_first(int m, int n)
{
   for (int i = 0; i < strlen(first[n]); i++)
   {
      int flag = 0;
      for (int j = 0; j < strlen(follow[m]); j++)
      {
         if (first[n][i] == follow[m][j])
            flag = 1;
      }
```

```
        if (flag == 0)
            add_to_follow(m, first[n][i]);
    }
}

void find_first(struct state *I)
{
    for (int i = 0; i < no_nt; i++)
    {
        for (int j = 0; j < I->prod_count; j++)
        {
            if (I->prod[j][0] == non_terminals[i])
            {
                if (!is_non_terminal(I->prod[j][3]))
                {
                    add_to_first(i, I->prod[j][3]);
                }
            }
        }
    }
}

void find_follow(struct state *I)
{
    for (int i = 0; i < no_nt; i++)
    {
        for (int j = 0; j < I->prod_count; j++)
```

```
        {
            for (int k = 3; k < strlen(I->prod[j]); k++)
            {
                if (I->prod[j][k] == non_terminals[i])
                {
                    if (I->prod[j][k + 1] != '\0')
                    {
                        if (!is_non_terminal(I->prod[j][k + 1]))
                        {
                            add_to_follow(i, I->prod[j][k + 1]);
                        }
                    }
                }
            }
        }
    }
}

int get_index(int *arr, int n)
{
    for (int i = 0; i < no_t; i++)
    {
        if (arr[i] == n)
            return i;
    }
    return -1;
}
```

```
void print_reduce_table(int state_count, int *no_re, struct state *temp1)
{
    cout << "**********Reduce actions**********" << endl
         << endl;
    cout << "\t";
    int arr[temp1->prod_count][no_t] = {-1};
    for (int i = 0; i < no_t; i++)
    {
        cout << terminals[i] << "\t";
    }
    cout << endl;
    for (int i = 0; i < temp1->prod_count; i++)
    {
        int n = no_re[i];
        for (int j = 0; j < strlen(follow[return_index(temp1->prod[i][0]) - no_t]);
j++)
        {
            for (int k = 0; k < no_t; k++)
            {
                if (follow[return_index(temp1->prod[i][0]) - no_t][j] == terminals[k])
                    arr[i][k] = i + 1;
            }
        }
        cout << "I" << n << "\t";
        for (int j = 0; j < no_t; j++)
        {
```

```
            if (arr[i][j] != -1 && arr[i][j] != 0 && arr[i][j] < state_count)
                cout << "R" << arr[i][j] << "\t";
            else
                cout << "\t";
        }
        cout << endl;
    }
}

int main()
{
    struct state init;
    struct state temp;
    struct state temp1;
    int state_count = 1;
    get_prods(&init);
    temp = init;
    temp1 = temp;
    add_dots(&init);

    for (int i = 0; i < 100; i++)
        for (int j = 0; j < 100; j++)
            goto_table[i][j] = '~';

    struct state I[50];
    augument(&I[0], &init);
    closure(&init, &I[0]);
```

```
    cout << "\nI0:\n";
    print_prods(&I[0]);

    char characters[20] = {};
    for (int i = 0; i < state_count; i++)
    {
        char characters[20] = {};
        for (int z = 0; z < I[i].prod_count; z++)
            if (!in_array(characters, char_after_dot(I[i].prod[z])))
                characters[strlen(characters)] = char_after_dot(I[i].prod[z]);

        for (int j = 0; j < strlen(characters); j++)
        {
            goto_state(&I[i], &I[state_count], characters[j]);
            closure(&init, &I[state_count]);
            int flag = 0;
            for (int k = 0; k < state_count - 1; k++)
            {
                if (same_state(&I[k], &I[state_count]))
                {
                    cleanup_prods(&I[state_count]);
                    flag = 1;
                    cout << "I" << i << " on reading the symbol " << characters[j] << "
goes to I" << k << ".\n";
                    goto_table[i][k] = characters[j];
                    ;
                    break;
```

```
            }
        }
        if (flag == 0)
        {
            state_count++;
            cout << "I" << i << " on reading the symbol " << characters[j] << " goes to I" << state_count - 1 << ":\n";
            goto_table[i][state_count - 1] = characters[j];
            print_prods(&I[state_count - 1]);
        }
    }
}

int no_re[temp.prod_count] = {-1};
terminals[no_t] = '$';
no_t++;

add_dot_at_end(&temp1);
for (int i = 0; i < state_count; i++)
{
    for (int j = 0; j < I[i].prod_count; j++)
        for (int k = 0; k < temp1.prod_count; k++)
            if (in_state(&I[i], temp1.prod[k]))
                no_re[k] = i;
}

find_first(&temp);
```

```
    for (int l = 0; l < no_nt; l++)
    {
        for (int i = 0; i < temp.prod_count; i++)
        {
            if (is_non_terminal(temp.prod[i][3]))
            {
                add_to_first(return_index(temp.prod[i][0]) - no_t,
return_index(temp.prod[i][3]) - no_t);
            }
        }
    }


    find_follow(&temp);
    add_to_follow(0, '$');
    for (int l = 0; l < no_nt; l++)
    {
        for (int i = 0; i < temp.prod_count; i++)
        {
            for (int k = 3; k < strlen(temp.prod[i]); k++)
            {
                if (temp.prod[i][k] == non_terminals[l])
                {
                    if (is_non_terminal(temp.prod[i][k + 1]))
                    {
                        add_to_follow_first(l, return_index(temp.prod[i][k + 1]) - no_t);
                    }
                    if (temp.prod[i][k + 1] == '\0')
```

```
                add_to_follow(l, return_index(temp.prod[i][0]) - no_t);
            }
        }
    }
    print_shift_table(state_count);
    cout << endl
        << endl;
    print_reduce_table(state_count, &no_re[0], &temp1);
}
```

OUTPUT:

```
Enter the number of productions:
3
Enter the number of non terminals:
2
Enter the non terminals one by one:
S
A
Enter the number of terminals:
2
Enter the terminals (single lettered) one by one:
e
f
Enter the productions one by one in form (S->ABc):
S->AA
A->eA
A->f

I0:
Z->.S
S->.AA
A->.eA
A->.f
```

Program 10

AIM: WAP in C to draw an operator precedence parsing table for the given grammar

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char stack[20],ip[20],opt[10][10][1],ter[10];
int i,j,k,n,top=0,col,row;
for(i=0;i<10;i++)
{
   stack[i]=NULL; ip[i]=NULL;
   for(j=0;j<10;j++)
   {
   opt[i][j][1]=NULL;

   }
}
printf("Enter the no.of terminals:");
scanf("%d",&n);
printf("\nEnter the terminals:");
scanf("%s",ter);
printf("\nEnter the table values:\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("Enter the value for %c %c:",ter[i],ter[j]);
scanf("%s",opt[i][j]);
```

```
}
}
printf("\nOPERATOR PRECEDENCE TABLE:\n");
for(i=0;i<n;i++){printf("\t%c",ter[i]);}
printf("\n");
for(i=0;i<n;i++){printf("\n%c",ter[i]);
for(j=0;j<n;j++){printf("\t%c",opt[i][j][0]);}}
stack[top]='$';
printf("\nEnter the input string:");
scanf("%s",ip);
i=0;
printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");
printf("\n%s\t\t\t%s\t\t\t",stack,ip);
while(i<=strlen(ip))
{
for(k=0;k<n;k++)
{
if(stack[top]==ter[k])
col=k;
if(ip[i]==ter[k])
row=k;
}
if((stack[top]=='$')&&(ip[i]=='$')){
printf("String is accepted");
break;}
else if((opt[col][row][0]=='<') ||(opt[col][row][0]=='='))
{ stack[++top]=opt[col][row][0];
stack[++top]=ip[i];
printf("Shift %c",ip[i]);
```

```
i++;
}
else
{
 if(opt[col][row][0]=='>')
{
while(stack[top]!='<'){--top;}
top=top-1;
printf("Reduce");
}
else
{
printf("\nString is not accepted");
break;
}
}
printf("\n");
for(k=0;k<=top;k++)
{
printf("%c",stack[k]);
}
printf("\t\t\t");
for(k=i;k<strlen(ip);k++){
printf("%c",ip[k]);
}
printf("\t\t\t");
}
getch();
}
```

OUTPUT:

```
Enter the no.of terminals:4

Enter the terminals:+*i$

Enter the table values:
Enter the value for + +:>
Enter the value for + *:<
Enter the value for + i:<
Enter the value for + $:>
Enter the value for * +:>
Enter the value for * *:>
Enter the value for * i:<
Enter the value for * $:>
Enter the value for i +:>
Enter the value for i *:>
Enter the value for i i:e
Enter the value for i $:>
Enter the value for $ +:<
Enter the value for $ *:<
Enter the value for $ i:<
Enter the value for $ $:a
```

```
OPERATOR PRECEDENCE TABLE:
        +       *       i       $

+       >       <       <       >
*       >       >       <       >
i       >       >       e       >
$       <       <       <       a
Enter the input string:i*i+i$

STACK                   INPUT STRING                    ACTION

$                       i*i+i$                  Shift i
$<i                     *i+i$                   Reduce
$                       *i+i$                   Shift *
$<*                     i+i$                    Shift i
$<*<i                   +i$                     Reduce
$<*                     +i$                     Reduce
$                       +i$                     Shift +
$<+                     i$                      Shift i
$<+<i                   $                       Reduce
$<+                     $                       Reduce
$                       $                       String is accepted
```

Program-11

AIM: WAP in C to draw a LL parsing table for a given grammar

PROGRAM:

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

void followfirst(char, int, int);
```

```
void findfirst(char, int, int);
void follow(char c);

int count, n = 0;
char calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10], first[10];
char f[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;
   printf("How many productions ? :");
   scanf("%d", &count);
   printf("\nEnter %d productions in form A=B where A and B are grammar
symbols :\n\n", count);
   for (i = 0; i < count; i++)
   {
      scanf("%s%c", production[i], &ch);
   }
```

```
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++;
   }
   char ter[10];
   for (k = 0; k < 10; k++)
   {
      ter[k] = '!';
   }
   int ap, vp, sid = 0;
   for (k = 0; k < count; k++)
   {
      for (kay = 0; kay < count; kay++)
      {
```

```
        if (!isupper(production[k][kay]) && production[k][kay] != '#' &&
production[k][kay] != '=' && production[k][kay] != '\0')
        {
           vp = 0;
           for (ap = 0; ap < sid; ap++)
           {
              if (production[k][kay] == ter[ap])
              {
                 vp = 1;
                 break;
              }
           }
           if (vp == 0)
           {
              ter[sid] = production[k][kay];
              sid++;
           }
        }
     }
  }
  ter[sid] = '$';
  sid++;
  printf("\n\t\t\t\t\t\t\t The LL(1) Parsing Table for the above grammer :-");

printf("\n\t\t\t\t\t\t\t^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^\n");

printf("\n\t\t\t=========================================
```

```
==========================================================
================\n");
   printf("\t\t\t\t|\t");
   for (ap = 0; ap < sid; ap++)
   {
      printf("%c\t\t", ter[ap]);
   }

printf("\n\t\t\t=========================================
==========================================================
================\n");
   char first_prod[count][sid];
   for (ap = 0; ap < count; ap++)
   {
      int destiny = 0;
      k = 2;
      int ct = 0;
      char tem[100];
      while (production[ap][k] != '\0')
      {
         if (!isupper(production[ap][k]))
         {
            tem[ct++] = production[ap][k];
            tem[ct++] = '_';
            tem[ct++] = '\0';
            k++;
            break;
         }
```

```
    else
    {
        int zap = 0;
        int tuna = 0;
        for (zap = 0; zap < count; zap++)
        {
            if (calc_first[zap][0] == production[ap][k])
            {
                for (tuna = 1; tuna < 100; tuna++)
                {
                    if (calc_first[zap][tuna] != '!')
                    {
                        tem[ct++] = calc_first[zap][tuna];
                    }
                    else
                        break;
                }
                break;
            }
        }
        tem[ct++] = '_';
    }
    k++;
}
int zap = 0, tuna;
for (tuna = 0; tuna < ct; tuna++)
{
```

```
      if (tem[tuna] == '#')
      {
         zap = 1;
      }
      else if (tem[tuna] == '_')
      {
         if (zap == 1)
         {
            zap = 0;
         }
         else
            break;
      }
      else
      {
         first_prod[ap][destiny++] = tem[tuna];
      }
   }
}
char table[land][sid + 1];
ptr = -1;
for (ap = 0; ap < land; ap++)
{
   for (kay = 0; kay < (sid + 1); kay++)
   {
      table[ap][kay] = '!';
   }
```

```
}
for (ap = 0; ap < count; ap++)
{
   ck = production[ap][0];
   xxx = 0;
   for (kay = 0; kay <= ptr; kay++)
      if (ck == table[kay][0])
         xxx = 1;
   if (xxx == 1)
      continue;
   else
   {
      ptr = ptr + 1;
      table[ptr][0] = ck;
   }
}
for (ap = 0; ap < count; ap++)
{
   int tuna = 0;
   while (first_prod[ap][tuna] != '\0')
   {
      int to, ni = 0;
      for (to = 0; to < sid; to++)
      {
         if (first_prod[ap][tuna] == ter[to])
         {
            ni = 1;
```

```
            }
        }
        if (ni == 1)
        {
            char xz = production[ap][0];
            int cz = 0;
            while (table[cz][0] != xz)
            {
                cz = cz + 1;
            }
            int vz = 0;
            while (ter[vz] != first_prod[ap][tuna])
            {
                vz = vz + 1;
            }
            table[cz][vz + 1] = (char)(ap + 65);
        }
        tuna++;
    }
}
for (k = 0; k < sid; k++)
{
    for (kay = 0; kay < 100; kay++)
    {
        if (calc_first[k][kay] == '!')
        {
            break;
```

```
        }
        else if (calc_first[k][kay] == '#')
        {
            int fz = 1;
            while (calc_follow[k][fz] != '!')
            {
                char xz = production[k][0];
                int cz = 0;
                while (table[cz][0] != xz)
                {
                    cz = cz + 1;
                }
                int vz = 0;
                while (ter[vz] != calc_follow[k][fz])
                {
                    vz = vz + 1;
                }
                table[k][vz + 1] = '#';
                fz++;
            }
            break;
        }
    }
}
for (ap = 0; ap < land; ap++)
{
    printf("\t\t\t   %c\t|\t", table[ap][0]);
```

```
      for (kay = 1; kay < (sid + 1); kay++)
      {
         if (table[ap][kay] == '!')
            printf("\t\t");
         else if (table[ap][kay] == '#')
            printf("%c=#\t\t", table[ap][0]);
         else
         {
            int mum = (int)(table[ap][kay]);
            mum -= 65;
            printf("%s\t\t", production[mum]);
         }
      }
      printf("\n");
      printf("\t\t\t----------------------------------------------------------------------
----------------------------------------");
      printf("\n");
   }
   int j;
   printf("\n\nPlease enter the desired INPUT STRING = ");
   char input[100];
   scanf("%s%c", input, &ch);

printf("\n\t\t\t\t\t========================================
==================================\n");
   printf("\t\t\t\t\t\tStack\t\t\tInput\t\t\tAction");

printf("\n\t\t\t\t\t========================================
==================================\n");
```

```
int i_ptr = 0, s_ptr = 1;
char stack[100];
stack[0] = '$';
stack[1] = table[0][0];
while (s_ptr != -1)
{
   printf("\t\t\t\t\t\t");
   int vamp = 0;
   for (vamp = 0; vamp <= s_ptr; vamp++)
   {
      printf("%c", stack[vamp]);
   }
   printf("\t\t\t");
   vamp = i_ptr;
   while (input[vamp] != '\0')
   {
      printf("%c", input[vamp]);
      vamp++;
   }
   printf("\t\t\t");
   char her = input[i_ptr];
   char him = stack[s_ptr];
   s_ptr--;
   if (!isupper(him))
   {
      if (her == him)
      {
```

```
        i_ptr++;
        printf("POP ACTION\n");
    }
    else
    {
        printf("\nString Not Accepted by LL(1) Parser !!\n");
        exit(0);
    }
}
else
{
    for (i = 0; i < sid; i++)
    {
        if (ter[i] == her)
            break;
    }
    char produ[100];
    for (j = 0; j < land; j++)
    {
        if (him == table[j][0])
        {
            if (table[j][i + 1] == '#')
            {
                printf("%c=#\n", table[j][0]);
                produ[0] = '#';
                produ[1] = '\0';
            }
```

```
            else if (table[j][i + 1] != '!')
            {
               int mum = (int)(table[j][i + 1]);
               mum -= 65;
               strcpy(produ, production[mum]);
               printf("%s\n", produ);
            }
            else
            {
               printf("\nString Not Accepted by LL(1) Parser !!\n");
               exit(0);
            }
         }
      }
      int le = strlen(produ);
      le = le - 1;
      if (le == 0)
      {
         continue;
      }
      for (j = le; j >= 2; j--)
      {
         s_ptr++;
         stack[s_ptr] = produ[j];
      }
   }
}
```

```
printf("\n\t\t\t==========================================
==========================================================
==================\n");
   if (input[i_ptr] == '\0')
   {
      printf("\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN ACCEPTED !!\n");
   }
   else
      printf("\n\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN REJECTED !!\n");

printf("\t\t\t==========================================
==========================================================
=================\n");
}


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)
         {
```

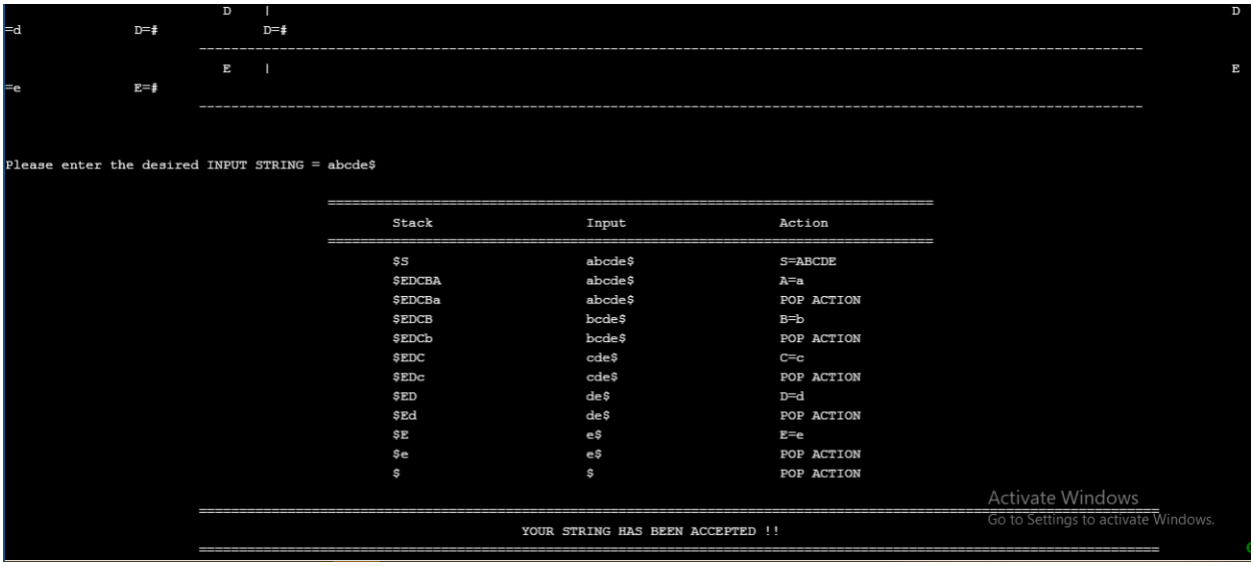
```
            if (production[i][j + 1] != '\0')
            {
               followfirst(production[i][j + 1], i, (j + 2));
            }
            if (production[i][j + 1] == '\0' && c != production[i][0])
            {
               follow(production[i][0]);
            }
         }
      }
   }
}

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:
```

```
How many productions ? :10

Enter 10 productions in form A=B where A and B are grammar symbols :

S=ABCDE
A=a
A=#
B=b
B=#
C=c
D=d
D=#
E=e
E=#

 First(S)= { a, b, c, }

 First(A)= { a, #, }

 First(B)= { b, #, }

 First(C)= { c, }

 First(D)= { d, #, }

 First(E)= { e, #, }

------------------------------------------------
```

```
 Follow(S) = { $,  }

 Follow(A) = { b, c,  }

 Follow(B) = { c,  }

 Follow(C) = { d, e, $,  }

 Follow(D) = { e, $,  }

 Follow(E) = { $,  }

                                        The LL(1) Parsing Table for the above grammer :-
                                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

                    ================================================================================================================
                           |       a               b               c                                                               $
                    ================================================================================================================
                      S    |       S=ABCDE         S=ABCDE         S=ABCDE
                    ----------------------------------------------------------------------------------------------------------------
                      A    |       A=a             A=#             A=#
                    ----------------------------------------------------------------------------------------------------------------
                      B    |                       B=b             B=#
                    ----------------------------------------------------------------------------------------------------------------
                      C    |                                       C=c
                    ----------------------------------------------------------------------------------------------------------------
                      D    |                                                                                                       D
=d              D=#             D=#
```


D |
=d D=# D=#
E |
=e E=#
Please enter the desired INPUT STRING = abcde$
Stack Input Action
$S abcde$ S=ABCDE
$EDCBA abcde$ A=a
$EDCBa abcde$ POP ACTION
$EDCB bcde$ B=b
$EDCb bcde$ POP ACTION
$EDC cde$ C=c
$EDc cde$ POP ACTION
$ED de$ D=d
$Ed de$ POP ACTION
$E e$ E=e
$e e$ POP ACTION
$ $ POP ACTION
YOUR STRING HAS BEEN ACCEPTED !!
Activate Windows
Go to Settings to activate Windows.