**Compiler Design Lab**

**(RCS-652)**

**Laboratory Manual**

For

**Bachelor of Technology**

In

**Information Technology**

**Odd Semester**

**NAME = SUMIT SINGH BISHT**

**ROLL NO= 1816110213**



**Department of Information Technology**

**KRISHNA ENGINEERING COLLEGE**

95, Loni Road, Mohan Nagar, Ghaziabad (Uttar Pradesh), Pin-201007

**Course Outcomes of Compiler Design LAB**

**(RCS-652)**

1. Abilty to create lexical rules and grammars for a programming language

2*.* Ability to use Flex or similar tools to create a lexical analyzer and Yacc/Bison tools to create a parser.

3. Ability to implement a lexer without using Flex or any other lexer generation tools.

4. Ability to implement a parser such as a bottom-up SLR parser without using Yacc/Bison or any other compiler-generation tools.

5. Ability to implement semantic rules into a parser that performs attribution while parsing.

6. Abilty to design a compiler for a concise programming language.

**KRISHNA ENGINEERING COLLEGE**

**Department of Computer Science & Engineering**

**List of Practical’s**

**COMPILER DESIGN LAB (RCS-652)**

|  |  |
| --- | --- |
| S.No. | LIST OF PROGRAMS |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | WAP to check whether the entered string is accepted or not for a given grammar.  WAP to convert infix expression to postfix expression.  WAP to convert infix expression to prefix expression.  WAP to find the no. of tokens and list them according to their category in an expression (given/entered)  WAP to construct an NFA from a regular expression (given) and display the transition table of NFA constructed.  WAP to compute LEADING and TRAILING sets of a grammar (given).  WAP to calculate FIRST and FOLLOW  WAP in C to check whether the Grammar is Left-recursive and remove left recursion  WAP in C to draw a SLR parsing table for a given grammar.  WAP in C to draw an operator precedence parsing table for the given grammar  WAP in C to draw a LL parsing table for a given grammar |

**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<conio.h>

#include<string.h>

#include<stdlib.h>

char a[100];

int n, i;

void main()

{

clrscr();

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’)

{

printf(“\n string is not accepted”);

getch();

exit(0);

}

}

printf(“\n string is accepted”);

}

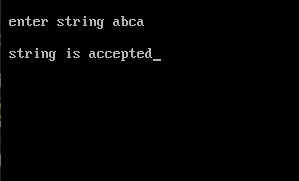
else

printf(“\n string is not accepted”);

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<string.h>

char str[]="A+(C\*D)\*F";

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 postfix[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!='(')

{

postfix[j]=temp;

j++;

temp=pop();

}

break;

case '+':

case '-':

case '\*':

case'/':

while(precede(stack[top])>=precede(s))

{

temp=pop();

postfix[j]=temp;

j++;

}

push(s);

break;

default:

postfix[j++]=s;

break;

}

i++;

}

while(top>0)

{

temp=pop();

postfix[j++]=temp;

}

postfix[j++]='\0';

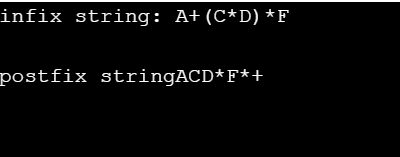
printf("\npostfix string");

puts(postfix);

getch();

}

**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) // Division(/)

return(5);

if(c==42) // Multiplication(\*)

return(4);

if(c==43) //Addition(+)

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");

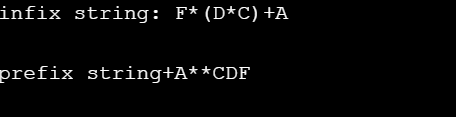
for(i=6;i>=0;i--)

printf("%c", prefix[i]);

getch();

}

**Output:**



**Program 4**

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

**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++;

}

/\* ASCII values:

/ -> 47

\* -> 42

+ -> 43

- -> 45

= -> 61

^ -> 94

\*/

void main()

{

char str[13];

char c;

clrscr();

printf("\nenter string");

scanf("%s", &str);

for(int i=0; i<13; i++)

{

c=str[i];

check(c);

}

for(i=0; i<13; i++)

{

if(isdigit(str[i])&&isdigit(str[i+1]))

{

i=i+2;

con++;

}

else if(isdigit(str[i]))

con++;

}

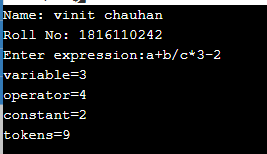
printf("\n operators: %d \nvariables: %d \nconstants: %d" , op, var, con);

printf("\ntotal tokens=%d", op+var+con);

getch();

}

**OUTPUT**:

****

**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:**

1. What is NFA

#include <iostream>

#include <conio.h>

#include <stdio.h>

#include <string.h>

using namespace std;

int main()

{

char s[10];

int n, init = 0, fin = 1;

cout << "Enter Regular Expression\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;

fin += 1;

if (s[i] == '/')

fin += 4;

}

char c = 238;

int 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;

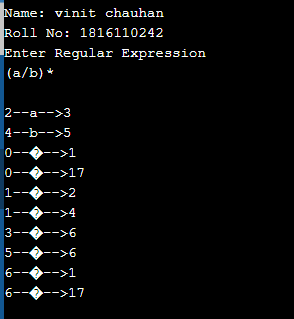
cout << "\n"

<< init + 6 << "--" << c << "-->" << fin;

return 0;

}

**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 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] << ",";

}

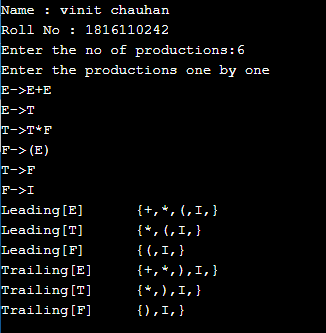
cout << "}\n";

}

return 0;

}

**Output:**



**Program 7**

**AIM: WAP to calculate FIRST and FOLLOW.**

**PROGRAM:**

FIRST

#include <stdio.h>

#include <string.h>

int i, j, l, m, n = 0, o, p, nv, z = 0, x = 0;

char str[10], temp, temp2[10], temp3[20], \*ptr;

struct prod

{

char lhs[10], rhs[10][10], ft[10], fol[10];

int n;

} pro[10];

void findter()

{

int k, t;

for (k = 0; k < n; k++)

{

if (temp == pro[k].lhs[0])

{

for (t = 0; t < pro[k].n; t++)

{

if (pro[k].rhs[t][0] < 65 || pro[k].rhs[t][0] > 90)

pro[i].ft[strlen(pro[i].ft)] = pro[k].rhs[t][0];

else if (pro[k].rhs[t][0] >= 65 && pro[k].rhs[t][0] <= 90)

{

temp = pro[k].rhs[t][0];

if (temp == 'S')

pro[i].ft[strlen(pro[i].ft)] = '#';

findter();

}

}

break;

}

}

}

void findfol()

{

int k, t, p1, o1, chk;

char \*ptr1;

for (k = 0; k < n; k++)

{

chk = 0;

for (t = 0; t < pro[k].n; t++)

{

ptr1 = strchr(pro[k].rhs[t], temp);

if (ptr1)

{

p1 = ptr1 - pro[k].rhs[t];

if (pro[k].rhs[t][p1 + 1] >= 65 && pro[k].rhs[t][p1 + 1] <= 90)

{

for (o1 = 0; o1 < n; o1++)

if (pro[o1].lhs[0] == pro[k].rhs[t][p1 + 1])

{

strcat(pro[i].fol, pro[o1].ft);

chk++;

}

}

else if (pro[k].rhs[t][p1 + 1] == '\0')

{

temp = pro[k].lhs[0];

if (pro[l].rhs[j][p] == temp)

continue;

if (temp == 'S')

strcat(pro[i].fol, "$");

findfol();

chk++;

}

else

{

pro[i].fol[strlen(pro[i].fol)] = pro[k].rhs[t][p1 + 1];

chk++;

}

}

}

if (chk > 0)

break;

}

}

int main()

{

FILE \*f;

for (i = 0; i < 10; i++)

pro[i].n = 0;

f = fopen("grammer.txt", "r");

while (!feof(f))

{

fscanf(f, "%s", pro[n].lhs);

if (n > 0)

{

if (strcmp(pro[n].lhs, pro[n - 1].lhs) == 0)

{

pro[n].lhs[0] = '\0';

fscanf(f, "%s", pro[n - 1].rhs[pro[n - 1].n]);

pro[n - 1].n++;

continue;

}

}

fscanf(f, "%s", pro[n].rhs[pro[n].n]);

pro[n].n++;

n++;

}

printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");

for (i = 0; i < n; i++)

for (j = 0; j < pro[i].n; j++)

printf("%s = %s\n", pro[i].lhs, pro[i].rhs[j]);

pro[0].ft[0] = '#';

for (i = 0; i < n; i++)

{

for (j = 0; j < pro[i].n; j++)

{

if (pro[i].rhs[j][0] < 65 || pro[i].rhs[j][0] > 90)

{

pro[i].ft[strlen(pro[i].ft)] = pro[i].rhs[j][0];

}

else if (pro[i].rhs[j][0] >= 65 && pro[i].rhs[j][0] <= 90)

{

temp = pro[i].rhs[j][0];

if (temp == 'S')

pro[i].ft[strlen(pro[i].ft)] = '#';

findter();

}

}

}

printf("\n\nFIRST\n");

for (i = 0; i < n; i++)

{

printf("\n%s = ", pro[i].lhs);

for (j = 0; j < strlen(pro[i].ft); j++)

{

for (l = j - 1; l >= 0; l--)

if (pro[i].ft[l] == pro[i].ft[j])

break;

if (l == -1)

printf("%c", pro[i].ft[j]);

}

}

for (i = 0; i < n; i++)

temp2[i] = pro[i].lhs[0];

pro[0].fol[0] = '$';

for (i = 0; i < n; i++)

{

for (l = 0; l < n; l++)

{

for (j = 0; j < pro[i].n; j++)

{

ptr = strchr(pro[l].rhs[j], temp2[i]);

if (ptr)

{

p = ptr - pro[l].rhs[j];

if (pro[l].rhs[j][p + 1] >= 65 && pro[l].rhs[j][p + 1] <= 90)

{

for (o = 0; o < n; o++)

if (pro[o].lhs[0] == pro[l].rhs[j][p + 1])

strcat(pro[i].fol, pro[o].ft);

}

else if (pro[l].rhs[j][p + 1] == '\0')

{

temp = pro[l].lhs[0];

if (pro[l].rhs[j][p] == temp)

continue;

if (temp == 'S')

strcat(pro[i].fol, "$");

findfol();

}

else

pro[i].fol[strlen(pro[i].fol)] = pro[l].rhs[j][p + 1];

}

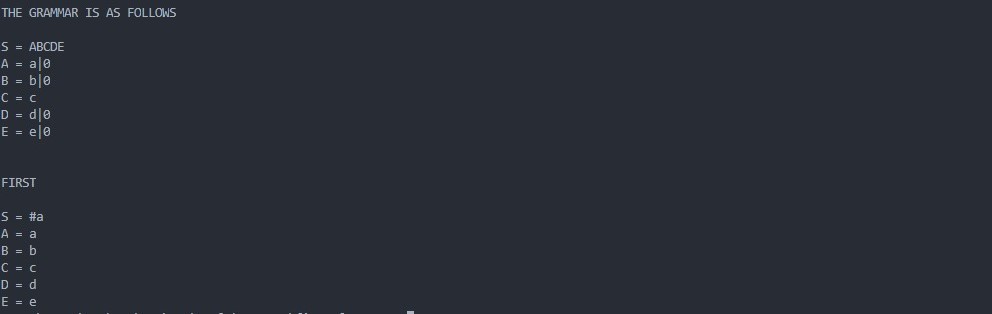
}

}

}

}

**Output: FIRST**

****

**FOLLOW**

#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;

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)

{

calc\_first[point1][point2++] = first[i];

}

}

jm = n;

point1++;

}

printf("\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);

}

}

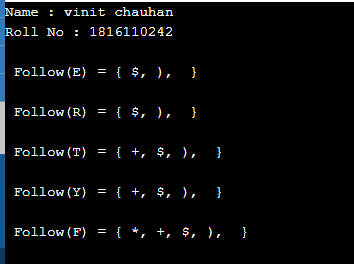
j++;

}

}

}

**OUTPUT: FOLLOW**



**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;

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

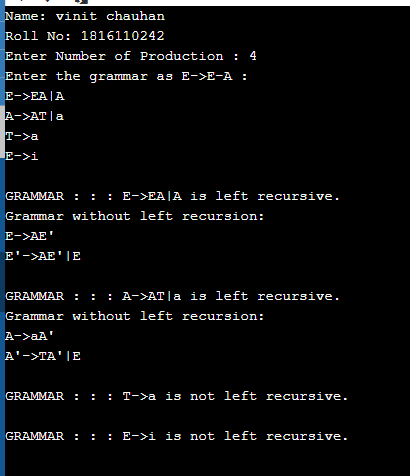
printf(" is not left recursive.\n");

index = 3;

}

}

**Output:**



**Program 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 (I0->prod\_count != I->prod\_count)

return false;

for (int i = 0; i < I0->prod\_count; i++)

{

int flag = 0;

for (int j = 0; j < I->prod\_count; j++)

if (strcmp(I0->prod[i], I->prod[j]) == 0)

flag = 1;

if (flag == 0)

return false;

}

return true;

}

void closure(struct state \*I, struct state \*I0)

{

char a = {};

for (int i = 0; i < I0->prod\_count; i++)

{

a = char\_after\_dot(I0->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()

{

cout<<"Name: utsav gaur "<<endl;

cout<<"Roll No: 1816110230"<<endl;

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);

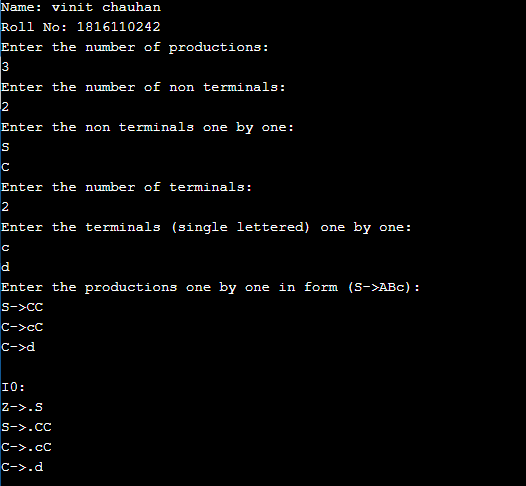
cout << endl

<< endl;

print\_reduce\_table(state\_count, &no\_re[0], &temp1);

}

**Output :**



**Program 10**

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

**PROGRAM:**

#include <iostream>

#include <string>

#include <deque>

using namespace std;

int n, n1, n2;

int getPosition(string arr[], string q, int size)

{

for (int i = 0; i < size; i++)

{

if (q == arr[i])

return i;

}

return -1;

}

int main()

{

string prods[10], leads[10], trails[10], nonterms[10], terms[10];

char op\_table[20][20] = {};

cout << "Enter the number of productions : ";

cin >> n;

cin.ignore();

cout << "Enter the productions" << endl;

for (int i = 0; i < n; i++)

{

getline(cin, prods[i]);

}

cout << "Enter the number of Terminals : ";

cin >> n2;

cin.ignore();

cout << "Enter the Terminals" << endl;

for (int i = 0; i < n2; i++)

{

cin >> terms[i];

}

terms[n2] = "$";

n2++;

cout << "Enter the number of Non-Terminals : ";

cin >> n1;

cin.ignore();

for (int i = 0; i < n1; i++)

{

cout << "Enter Non-Terminal : ";

getline(cin, nonterms[i]);

cout << "Enter Leads of " << nonterms[i] << " : ";

getline(cin, leads[i]);

cout << "Enter Trails of " << nonterms[i] << " : ";

getline(cin, trails[i]);

}

cout << "Enter the Rules (exit to stop)" << endl;

string rule = "";

while (rule != "exit")

{

getline(cin, rule);

if (rule[0] == '1')

{

int row = getPosition(terms, rule.substr(2, 1), n2);

int column = getPosition(terms, rule.substr(4, 1), n2);

op\_table[row][column] = '=';

}

if (rule[0] == '2')

{

int ntp = getPosition(nonterms, rule.substr(4, 1), n1);

int row = getPosition(terms, rule.substr(2, 1), n2);

for (int j = 0; j < leads[ntp].size(); j++)

{

int col = getPosition(terms, leads[ntp].substr(j, 1), n2);

op\_table[row][col] = '<';

}

}

if (rule[0] == '3')

{

int col = getPosition(terms, rule.substr(4, 1), n2);

int ntp = getPosition(nonterms, rule.substr(2, 1), n1);

for (int j = 0; j < trails[ntp].size(); j++)

{

int row = getPosition(terms, trails[ntp].substr(j, 1), n2);

op\_table[row][col] = '>';

}

}

}

for (int j = 0; j < leads[0].size(); j++)

{

int col = getPosition(terms, leads[0].substr(j, 1), n2);

op\_table[n2 - 1][col] = '<';

}

for (int j = 0; j < trails[0].size(); j++)

{

int row = getPosition(terms, trails[0].substr(j, 1), n2);

op\_table[row][n2 - 1] = '>';

}

cout << endl;

cout << "Grammar" << endl;

for (int i = 0; i < n; i++)

{

cout << prods[i] << endl;

}

//Display Table

for (int j = 0; j < n2; j++)

cout << "\t" << terms[j];

cout << endl;

for (int i = 0; i < n2; i++)

{

cout << terms[i] << "\t";

for (int j = 0; j < n2; j++)

{

cout << op\_table[i][j] << "\t";

}

cout << endl;

}

//Parsing String

char c;

do

{

string ip;

deque<string> op\_stack;

op\_stack.push\_back("$");

cout << "Enter the string to be parsed : ";

getline(cin, ip);

ip.push\_back('$');

cout << "Stack\ti/p Buffer\tRelation\tAction" << endl;

while (true)

{

for (int i = 0; i < op\_stack.size(); i++)

cout << op\_stack[i];

cout << "\t";

cout << ip << "\t";

int row = getPosition(terms, op\_stack.back(), n2);

int column = getPosition(terms, ip.substr(0, 1), n2);

if (op\_table[row][column] == '<')

{

op\_stack.push\_back("<");

op\_stack.push\_back(ip.substr(0, 1));

ip = ip.substr(1);

cout << "\t"

<< "<\t\tPush";

}

else if (op\_table[row][column] == '=')

{

op\_stack.push\_back("=");

op\_stack.push\_back(ip.substr(0, 1));

ip = ip.substr(1);

cout << "\t"

<< "=\t\tPush";

}

else if (op\_table[row][column] == '>')

{

string last;

do

{

op\_stack.pop\_back();

last = op\_stack.back();

op\_stack.pop\_back();

} while (last != "<");

cout << "\t"

<< ">\t\tPop";

}

else

{

if (ip[0] == '$' && op\_stack.back() == "$")

{

cout << "\t\t\tAccept\nString Parsed." << endl;

break;

}

else

{

cout << endl

<< "String cannot be Parsed." << endl;

break;

}

}

cout << endl;

}

cout << "Continue?(Y/N) ";

cin >> c;

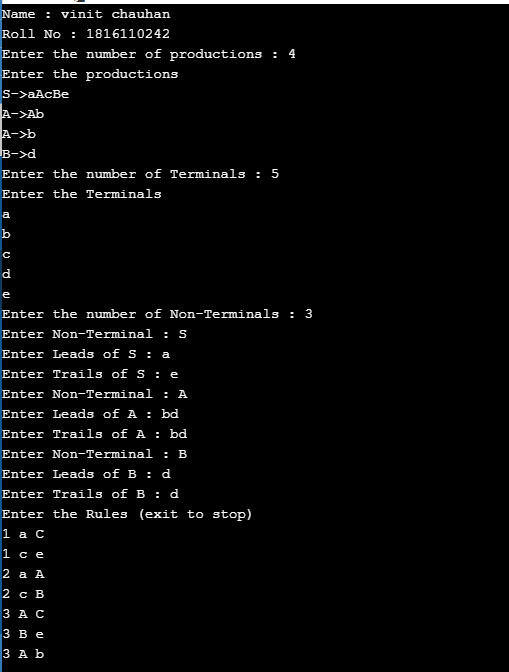
cin.ignore();

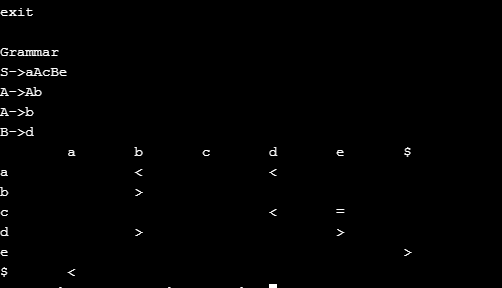
} while (c == 'y' || c == 'Y');

return 0;

}

**OUTPUT:**

****

****

**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:**

