# **Compiler Design Lab**

**(RCS-652)**

**Name : Ranjan Giri**

**Roll no : 1816110156**

**B.Tech -3rd year- Section-3C**

**5th Semester**

****

**Department of Information Technology**

**KRISHNA ENGINEERING COLLEGE**

95, Loni Road, Mohan Nagar, Ghaziabad (Uttar Pradesh), Pin-2010

**Program 1**

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

**PROGRAM:**

#include <iostream>

#include <string.h>

#include <stdlib.h>

using namespace std;

char a[100];

int n, i;

int main()

{

    cout << "\nenter string: ";

    cin >> 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')

            {

                cout << "string is not accepted";

                exit(0);

            }

        }

        cout << "string is accepted";

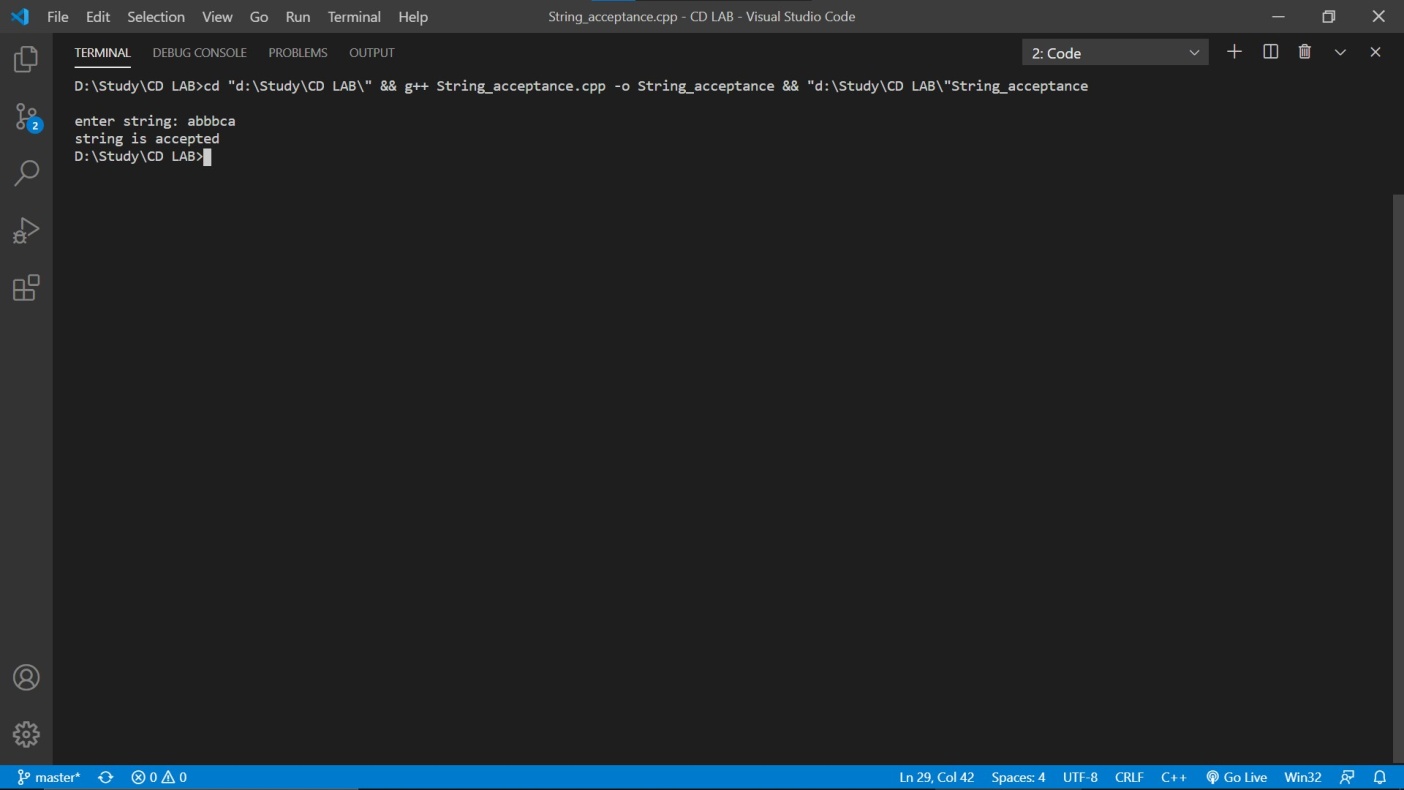
    }

    else

        cout << "string is not accepted";

}

**Output:**



**Program 2**

**AIM: WAP to convert infix expression to postfix expression.**

**PROGRAM:**

#include<stdio.h>

#include <ctype.h>

char stack[50];

int top=-1;

void push(char item) {

    stack[++top] = item;

}

char pop() {

    if (top==-1) {

        return -1;

    }

    else {

        return stack[top--];

    }

}

int priority(char p)

{

    if (p == '(')

        return 0;

    if (p == '+' || p == '-')

        return 1;

    if (p == '\*' || p == '/')

        return 2;

}

int main() {

    char i, str[50];

    printf("Enter infix expression:");

    scanf(" %s", str);

    char \*ptr=str;

    while (\*ptr != '\0') {

        if (isalnum(\*ptr)) {

            printf("%c", \*ptr);

        }

        else if (\*ptr == '(')

            push(\*ptr);

        else if (\*ptr == ')')

        {

            while ((i = pop()) != '(')

                printf("%c", i);

        }

        else

        {

            while (priority(stack[top]) >= priority(\*ptr)) {

                printf("%c", pop());

            }

            push(\*ptr);

        }

        ptr++;

    }

    while (top != -1)

    {

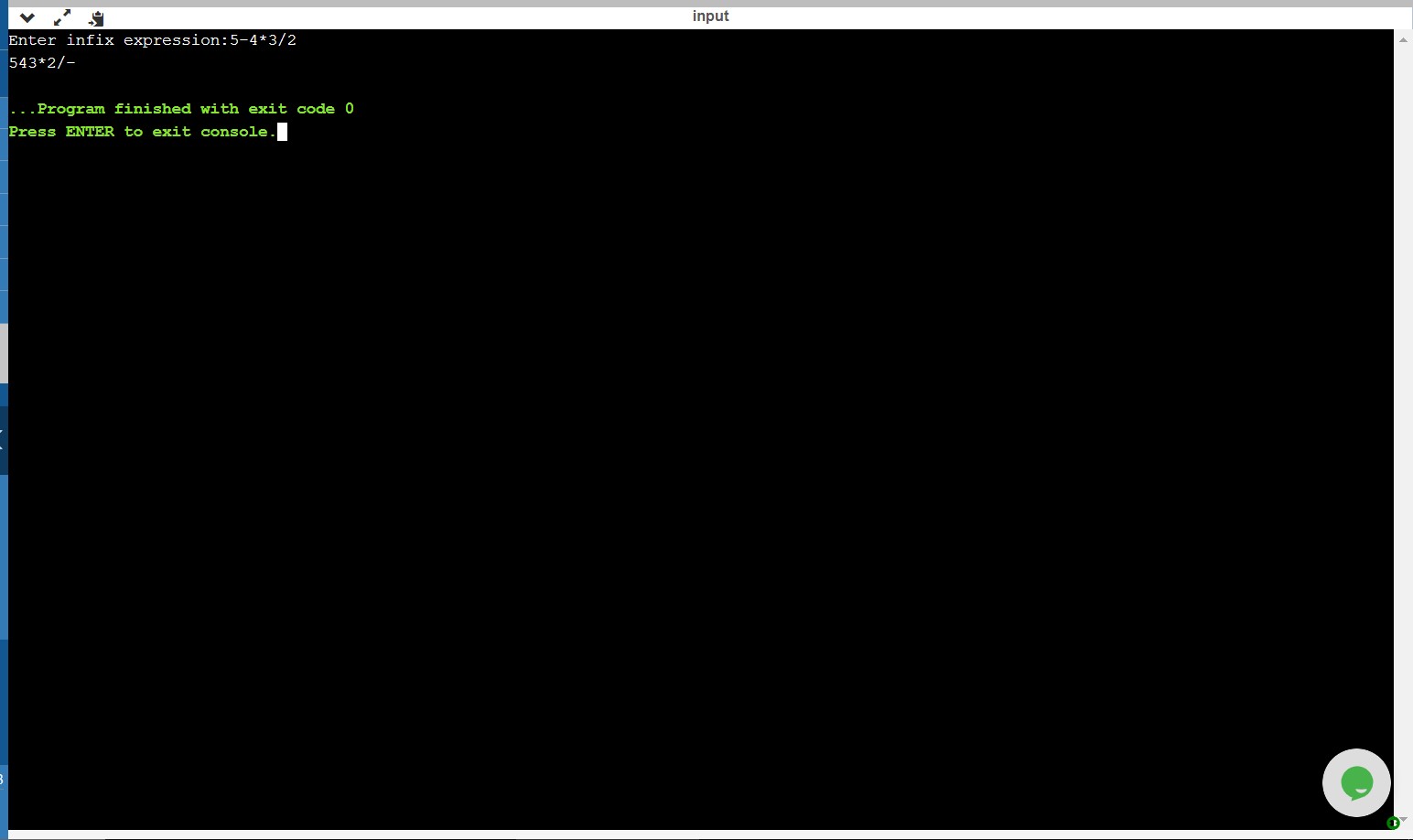
        printf("%c", pop());

    }

    return 0;

}

Output:



**Program 3**

**AIM: WAP to convert infix expression to prefix expression.**

**PROGRAM:**

#include <stdio.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);

}

int 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 : ");

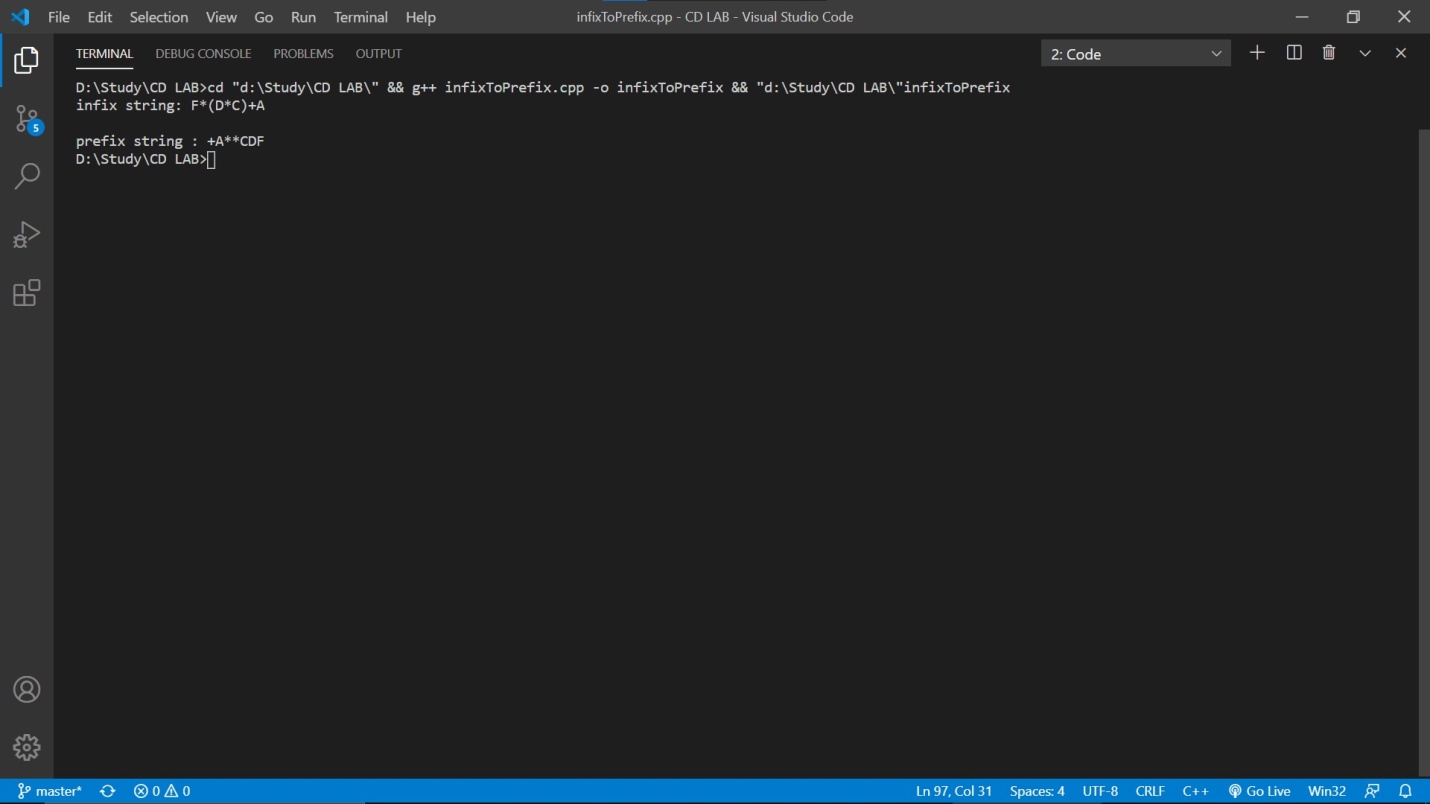
    for (i = 6; i >= 0; i--)

        printf("%c", prefix[i]);

    return 0;

}

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 <ctype.h>

int main()

{

    char i, str[50];

    int opr = 0, var = 0, con = 0;

    printf("Enter expression:");

    scanf(" %s", str);

    char \*ptr = str;

    while (\*ptr != '\0')

    {

        if (isalpha(\*ptr))

        {

            var++;

            while (isalnum(\*ptr))

                ptr++;

        }

        else if (!isalnum(\*ptr))

        {

            opr++;

            while (!isalnum(\*ptr))

                ptr++;

        }

        else if (isdigit(\*ptr))

        {

            con++;

            while (isdigit(\*ptr))

                ptr++;

        }

    }

    printf("variable=%d\n", var);

    printf("operator=%d\n", opr);

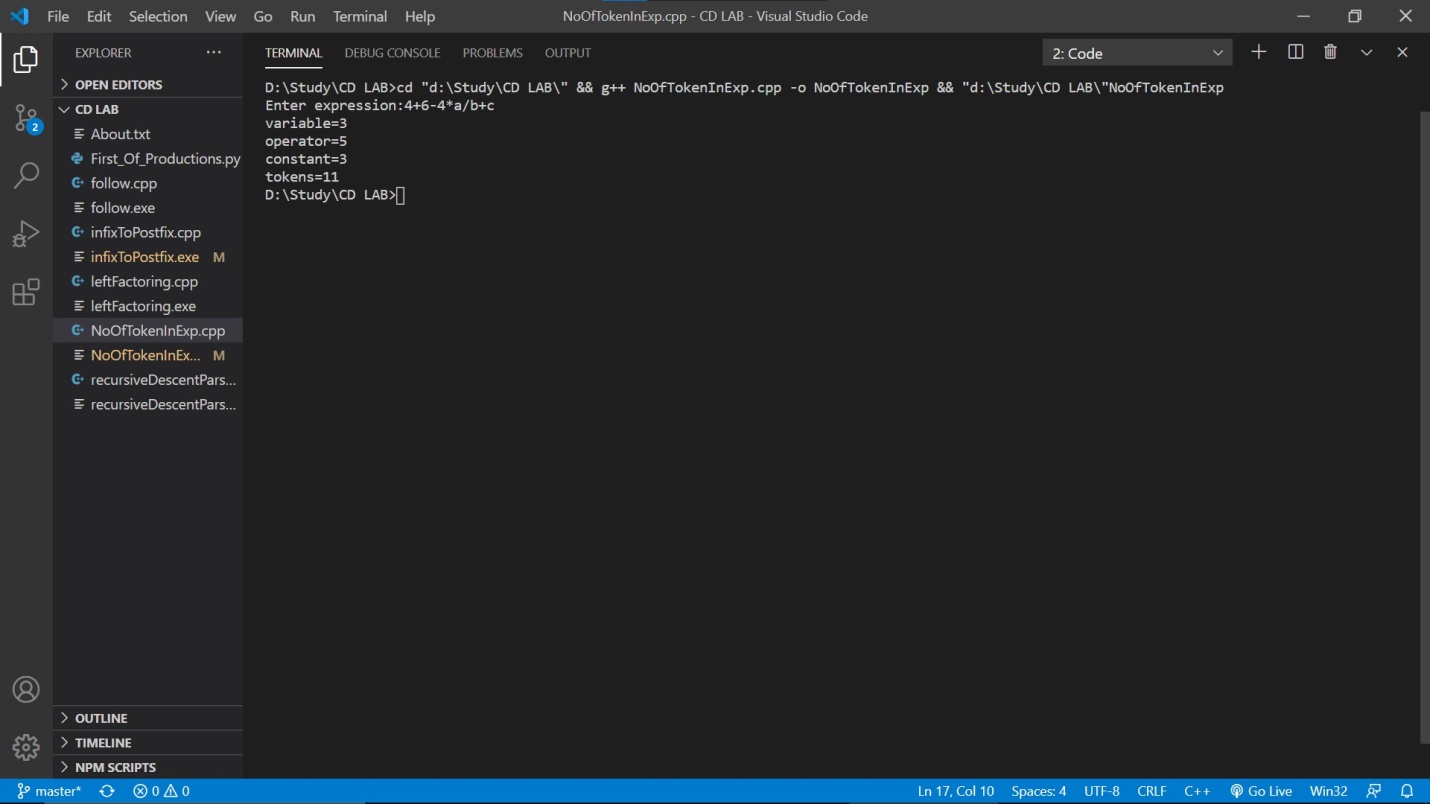
    printf("constant=%d\n", con);

    printf("tokens=%d", var + opr + con);

    return 0;

}

Output:



**Program 5**

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

**PROGRAM:**

#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 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;

        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;

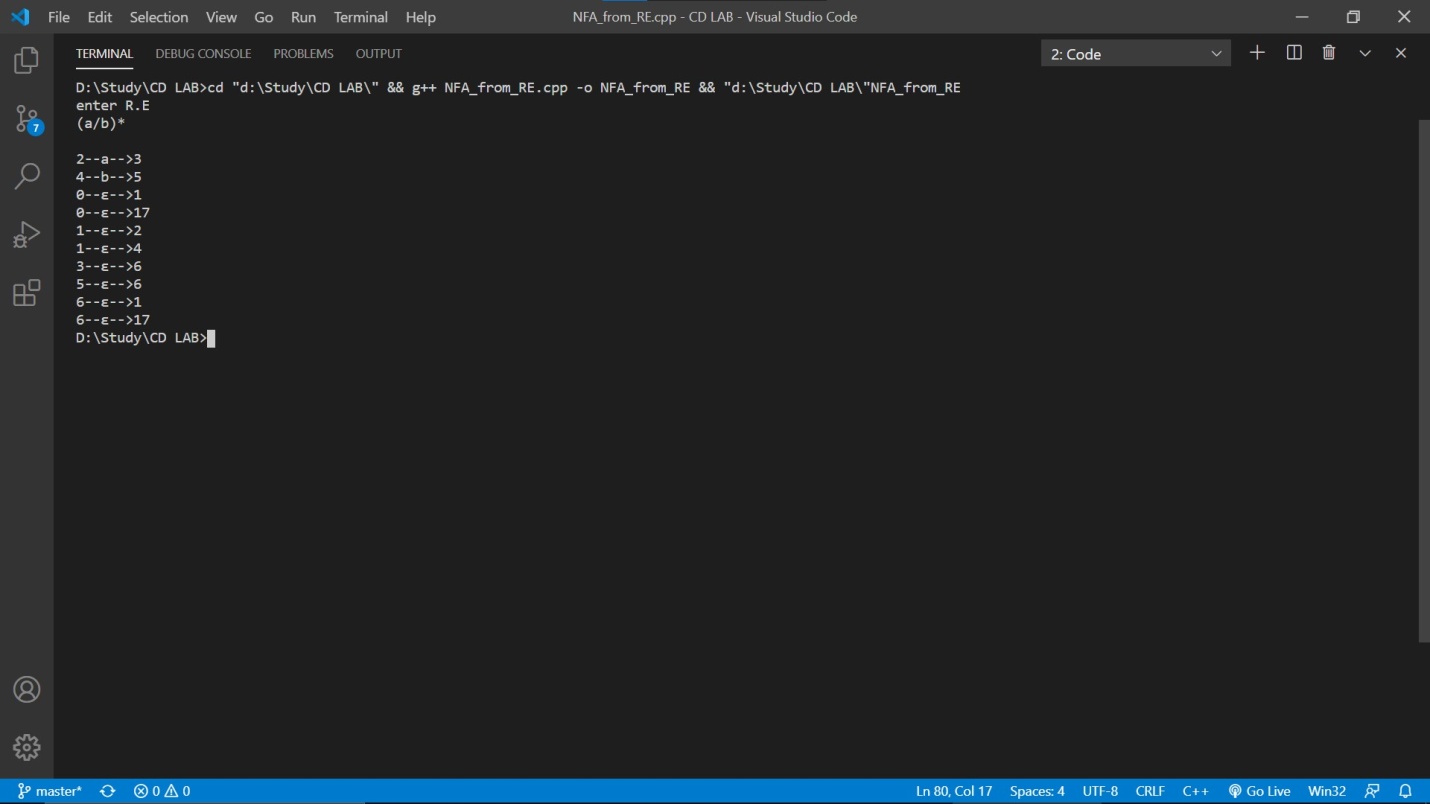
    cout << "\n"

         << 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>

using namespace std;

int main()

{

    char s, l[20], r[10], lead[10], trail[10];

    int n, j, m;

    for (int i = 0; i < 10; i++)

    {

        lead[i] = NULL;

        trail[i] = NULL;

    }

    cout << "\nenter total no. of productions";

    cin >> n;

    int k = 0;

    m = 0;

    for (int i = 0; i < n; i++)

    {

        cout << "\nenter the LHS of production";

        cin >> l[i];

        cout << "\nenter the RHS of production";

        cin >> r;

        for (int j = 0; j < 2; j++)

        {

            if ((r[j] == '(') || r[j] == ')' || r[j] == '\*' || r[j] == '+' || r[j] == '-' || r[j] == '/')

            {

                lead[k] = r[j];

                k = k + 1;

            }

            if ((r[j] == 'i') && (r[j + 1] == 'd'))

            {

                lead[k] = r[j];

                lead[k + 1] = r[j + 1];

                k = k + 1;

            }

        }

        for (j = 1; j <= 2; j++)

        {

            if ((r[j] == '(') || r[j] == ')' || r[j] == '\*' || r[j] == '+' || r[j] == '-' || r[j] == '/')

            {

                trail[m] = r[j];

                m = m + 1;

            }

            if ((r[j - 1] == 'i') && (r[j] == 'd'))

            {

                trail[m] = r[j - 1];

                trail[m + 1] = r[j];

                m = m + 1;

            }

        }

    }

    cout << "\nthe Leading(A) is :\n";

    cout << "{  ";

    for (int i = 0; i < k; i++)

    {

        if ((lead[i] == 'i') && (lead[i + 1] == 'd'))

            cout << lead[i] << lead[i + 1] << "  ";

        else

            cout << lead[i] << "  ";

    }

    cout << "}";

    cout << "\nthe Trailing(A) is :\n";

    cout << "{  ";

    for (int i = 0; i < m; i++)

    {

        if ((trail[i] == 'i') && (trail[i + 1] == 'd'))

            cout << trail[i] << trail[i + 1] << "  ";

        else

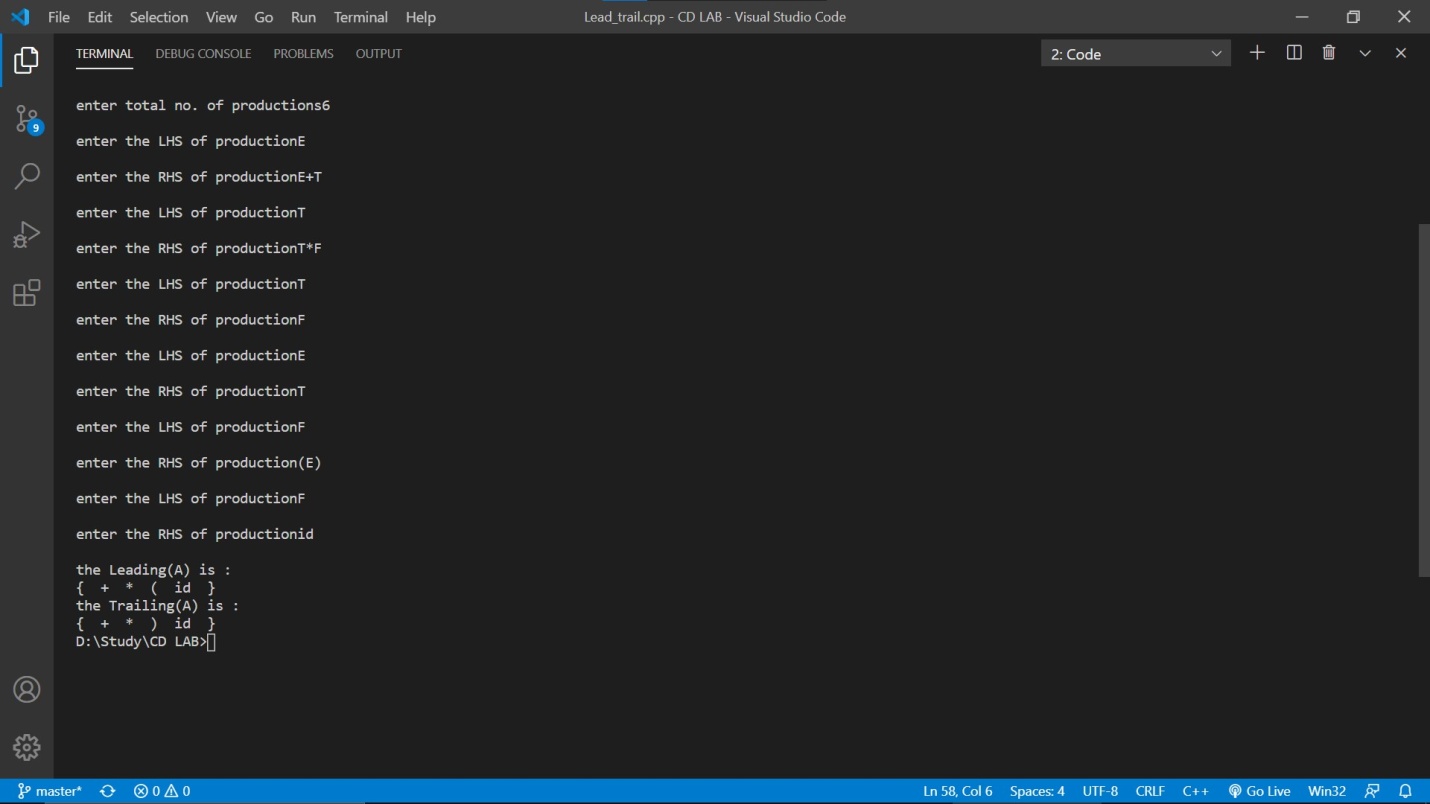
            cout << trail[i] << "  ";

    }

    cout << "}";

}

Output:



**Program 8**

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

**PROGRAM:**

def FIR(str):

    try:

        if(not str[2].isupper()):

            first[str[0]].append(str[2])

            # return first[str[0]]

        else:

            for i in productions:

                if(i[0] == str[2]):

                    FIR(str[0]+i[1:])

                    if('$' in first[str[0]]):

                        FIR(str[0:2]+str[3:])

    except:

        pass

productions = []

n = int(input("How many productions?"))

first = {}

for i in range(n):

    productions.append(input())

    str = productions[i]

    if (str[0] not in first):

        first[str[0]] = []

    FIR(str)

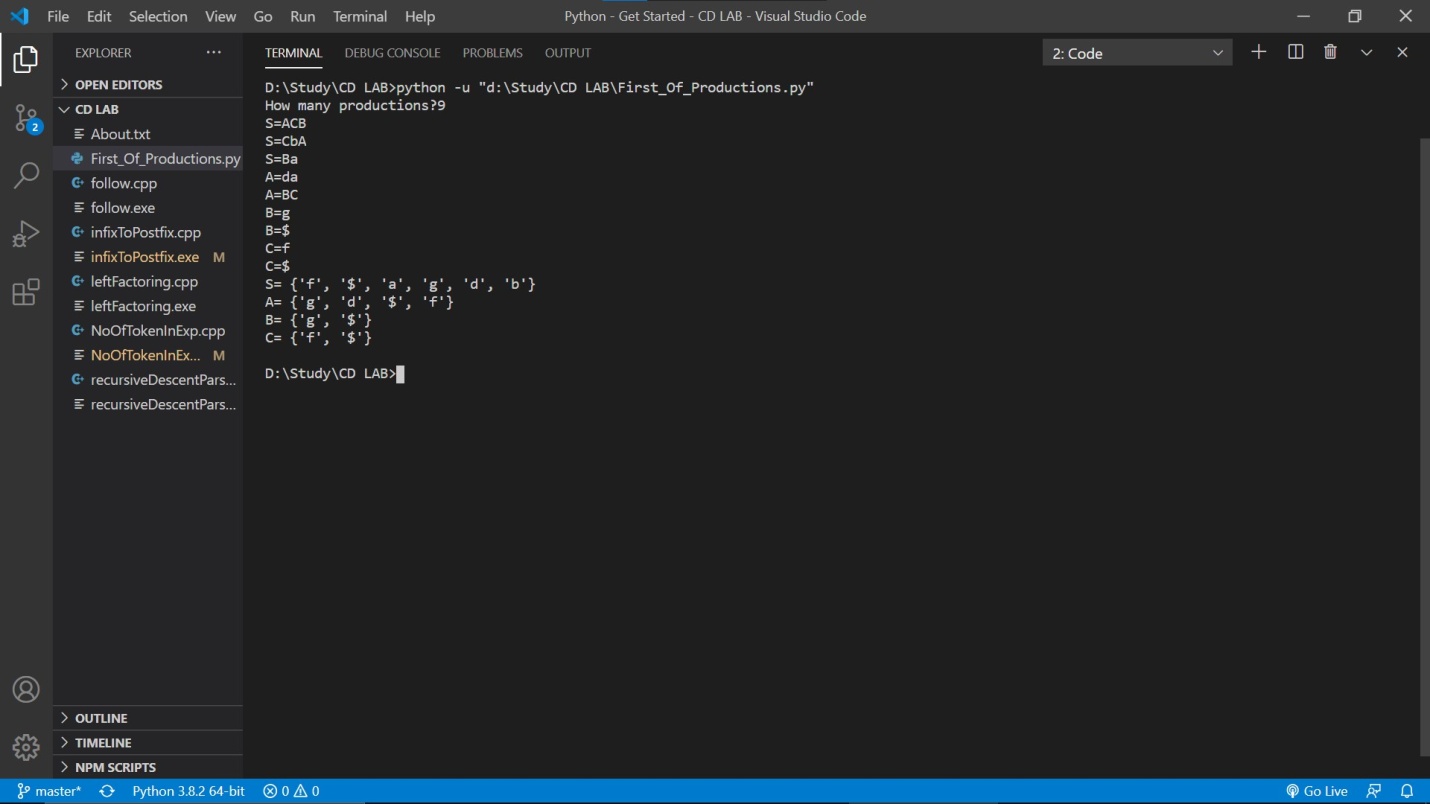
for i in productions:

    FIR(i)

for i in first:

    print(i+"=", set(first[i]))

Output:



Program:

#include <stdio.h>

#include <string.h>

#include <cctype>

int n, m = 0, p, i = 0, j = 0;

char a[10][10], f[10];

void follow(char c);

void first(char c);

int main()

{

    int i, z;

    char c, ch;

    printf("Enter the no.of productions:");

    scanf("%d", &n);

    printf("Enter the productions(epsilon=$):\n");

    for (i = 0; i < n; i++)

        scanf("%s%c", a[i], &ch);

    do

    {

        m = 0;

        printf("Enter the element whose FOLLOW is to be found:");

        scanf("%c", &c);

        follow(c);

        printf("FOLLOW(%c) = { ", c);

        for (i = 0; i < m; i++)

            printf("%c ", f[i]);

        printf(" }\n");

        printf("Do you want to continue(0/1)?");

        scanf("%d%c", &z, &ch);

    } while (z == 1);

}

void follow(char c)

{

    if (a[0][0] == c)

        f[m++] = '$';

    for (i = 0; i < n; i++)

    {

        for (j = 2; j < strlen(a[i]); j++)

        {

            if (a[i][j] == c)

            {

                if (a[i][j + 1] != '\0')

                    first(a[i][j + 1]);

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

                    follow(a[i][0]);

            }

        }

    }

}

void first(char c)

{

    int k;

    if (!(isupper(c)))

        f[m++] = c;

    for (k = 0; k < n; k++)

    {

        if (a[k][0] == c)

        {

            if (a[k][2] == '$')

                follow(a[i][0]);

            else if (islower(a[k][2]))

                f[m++] = a[k][2];

            else

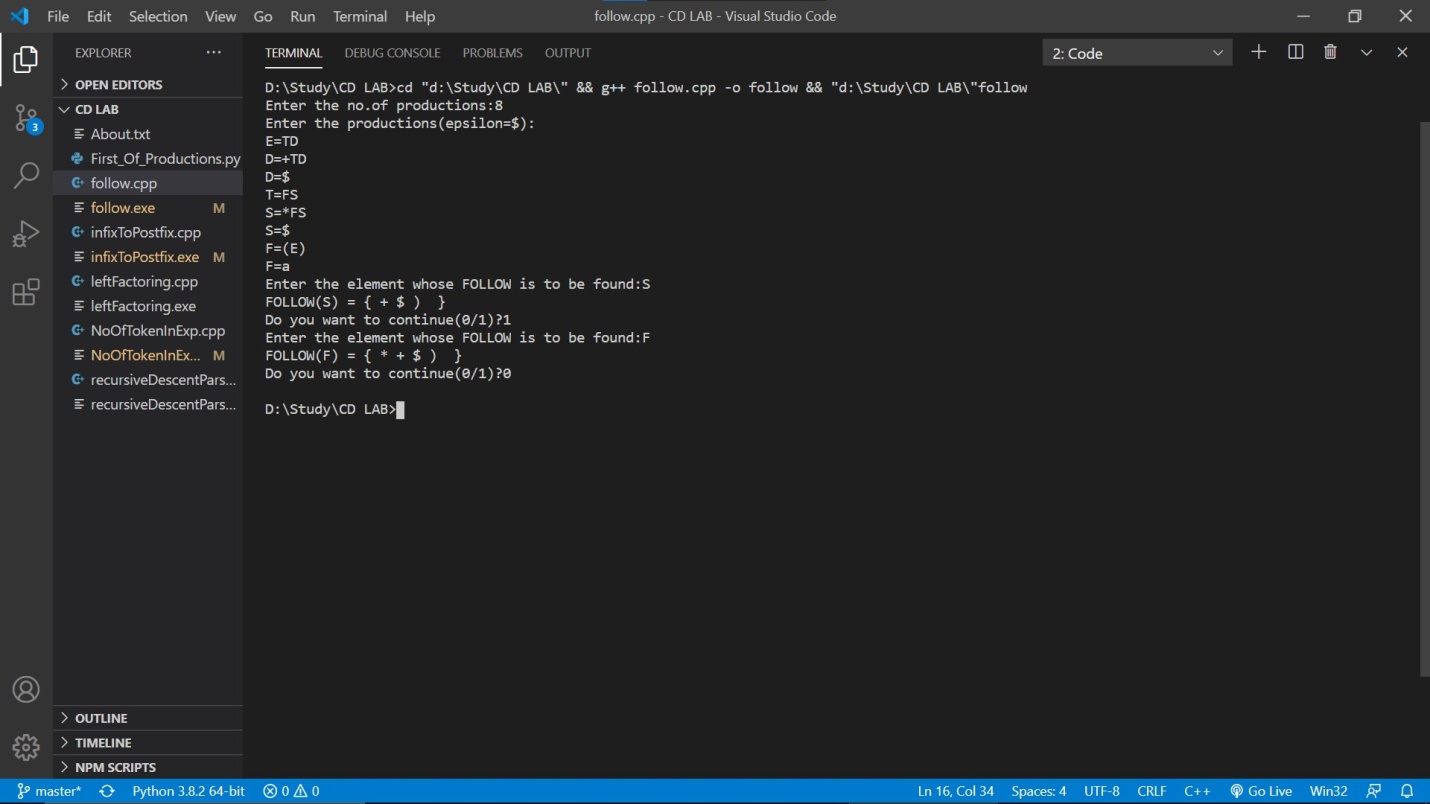
                first(a[k][2]);

        }

    }

}

Output:



**Program 9**

**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

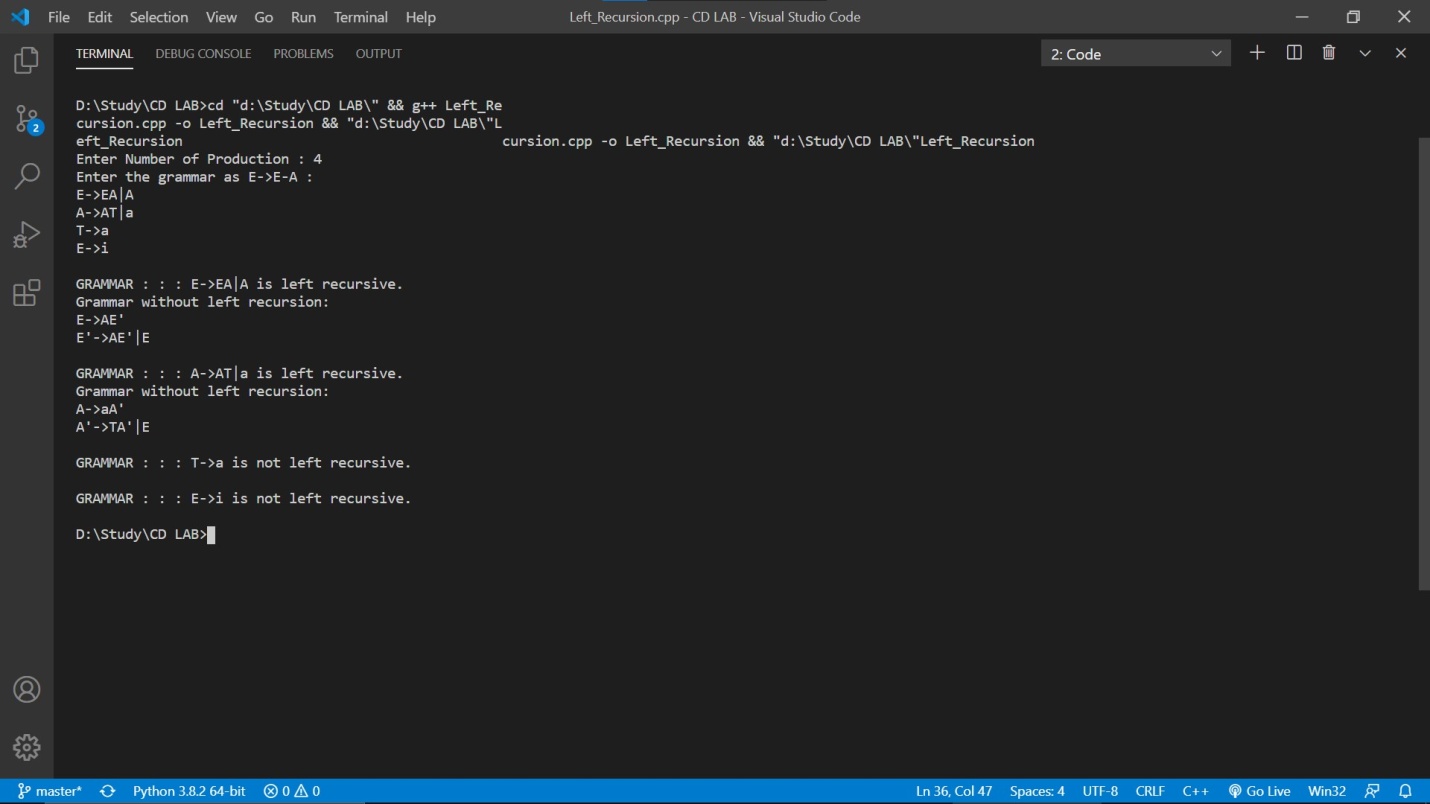
            printf(" is not left recursive.\n");

        index = 3;

    }

}

Output:



**Prgramm-10**

**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()

{

    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 11**

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

