1. Write a c program to find the given string is identifier or not.

Input

#include <stdio.h>

#include <string.h>

int main()

{

    char string[30];

    printf("Enter the string : ");

    scanf("%s",&string);

    if ((string[0] >=97 && string[0] <=122))

    {

        printf("The given string is a valid identifer...");

    }

    else

    {

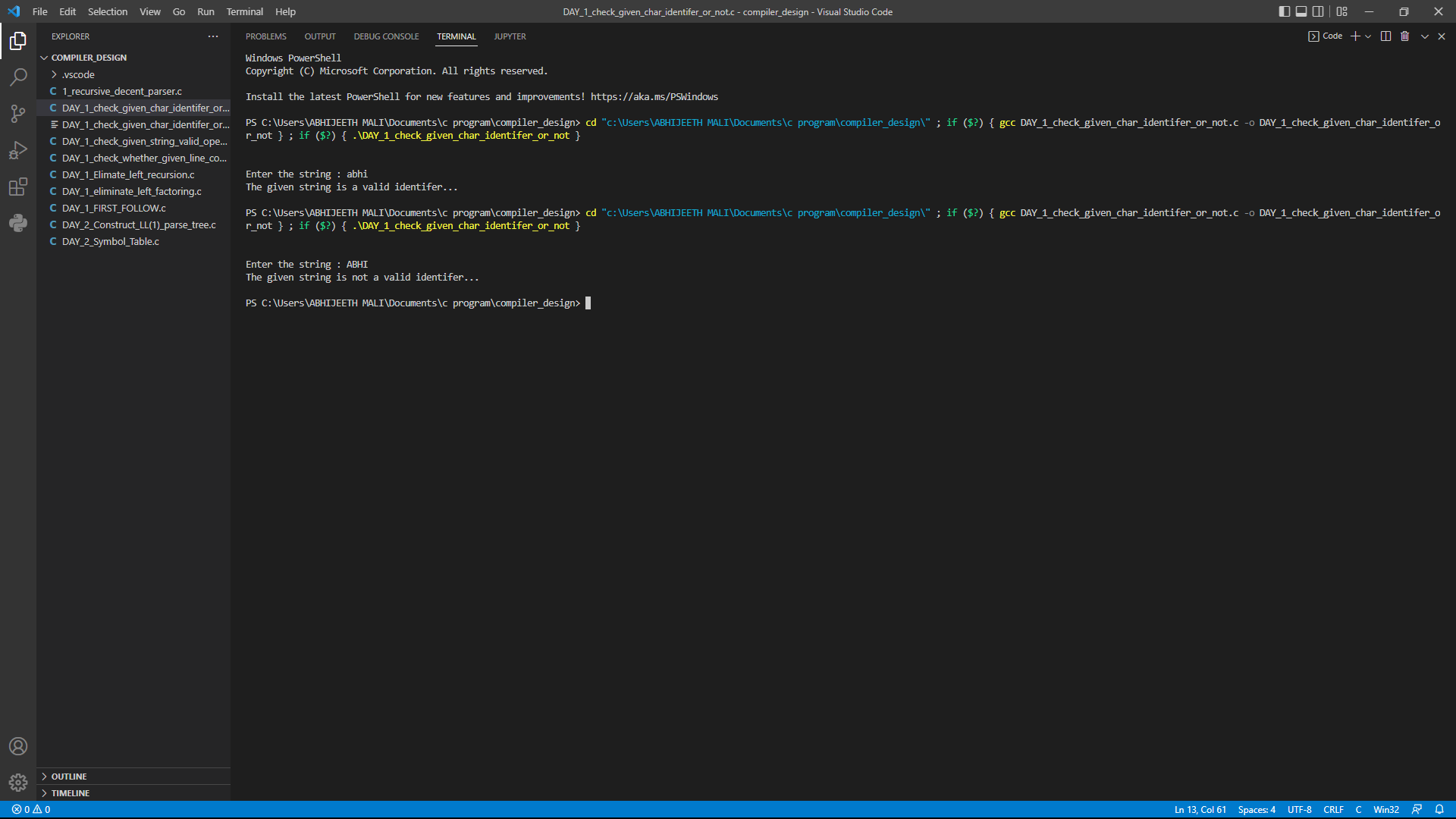
        printf("The given string is not a valid identifer...");

    }

    return 0;

}

Out put :



2. write a c program to find given input is valid operator or not

Input :

#include <stdio.h>

#include <string.h>

int main()

{

    char arithmetic[5] = {'+', '-', '\*', '/', '%'};

    char relational[4] = {'<', '>', '!', '='};

    char bitwise[5] = {'&', '^', '~', '|'};

    char str[2] = {' ', ' '};

    printf("\n\nEnter value to be identified: ");

    scanf("%s", &str);

    int i;

    if (((str[0] == '&' || str[0] == '|') && str[0] == str[1]) || (str[0] == '!' && str[1] == '\0'))

    {

        printf("\nIt is Logical operator");

    }

    for (i = 0; i < 4; i++)

    {

        if (str[0] == relational[i] && (str[1] == '=' || str[1] == '\0'))

        {

            printf("\n It is releational Operator\n\n");

            break;

        }

    }

    for (i = 0; i < 4; i++)

    {

        if ((str[0] == bitwise[i] && str[1] == '\0') || ((str[0] == '<' || str[0] == '>') && str[1] == str[0]))

        {

            printf("\n It is Bitwise Operator\n\n");

            break;

        }

    }

    if (str[0] == '?' && str[1] == ':')

        printf("\nIt  is  ternary operator\n\n");

    for (i = 0; i < 5; i++)

    {

        if ((str[0] == '+' || str[0] == '-') && str[0] == str[1])

        {

            printf("\nIt is unary operator\n\n");

            break;

        }

        else if ((str[0] == arithmetic[i] && str[1] == '=') || (str[0] == '=' && str[1] == ' '))

        {

            printf("\nIt is Assignment operator\n\n");

            break;

        }

        else if (str[0] == arithmetic[i] && str[1] == '\0')

        {

            printf("\nIt is arithmetic operator\n\n");

            break;

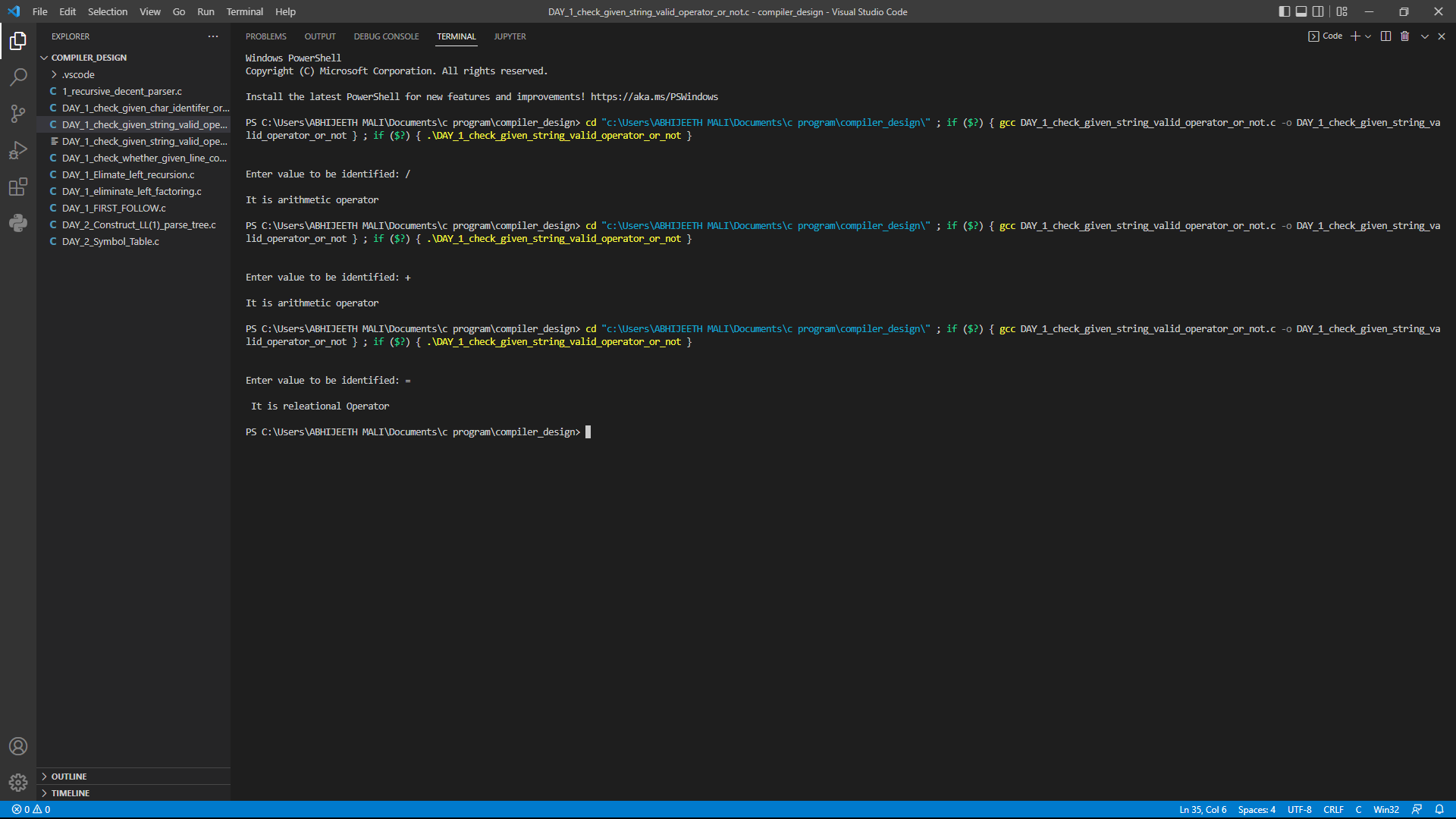
        }

    }

    return 0;

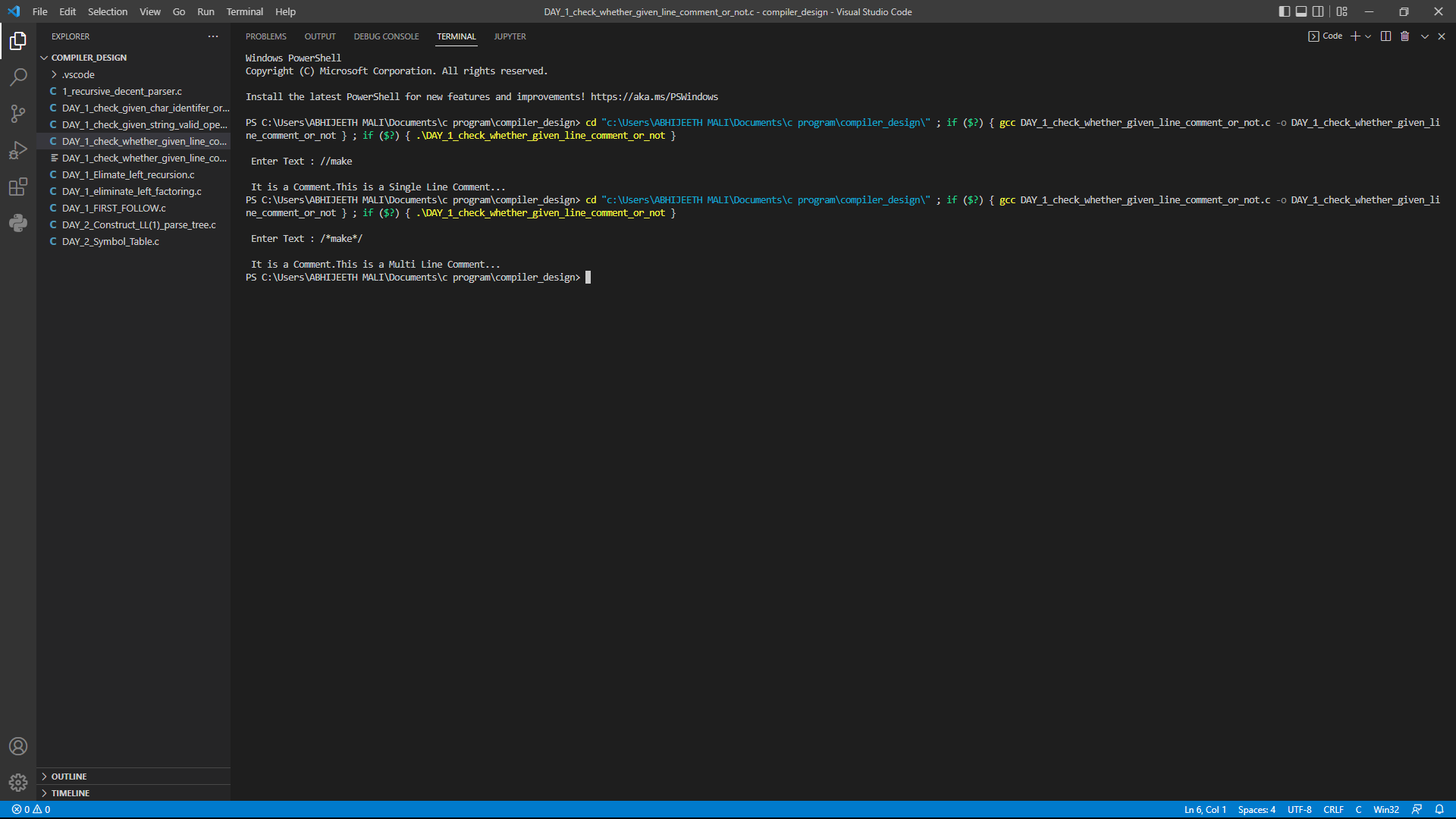
}

Output :

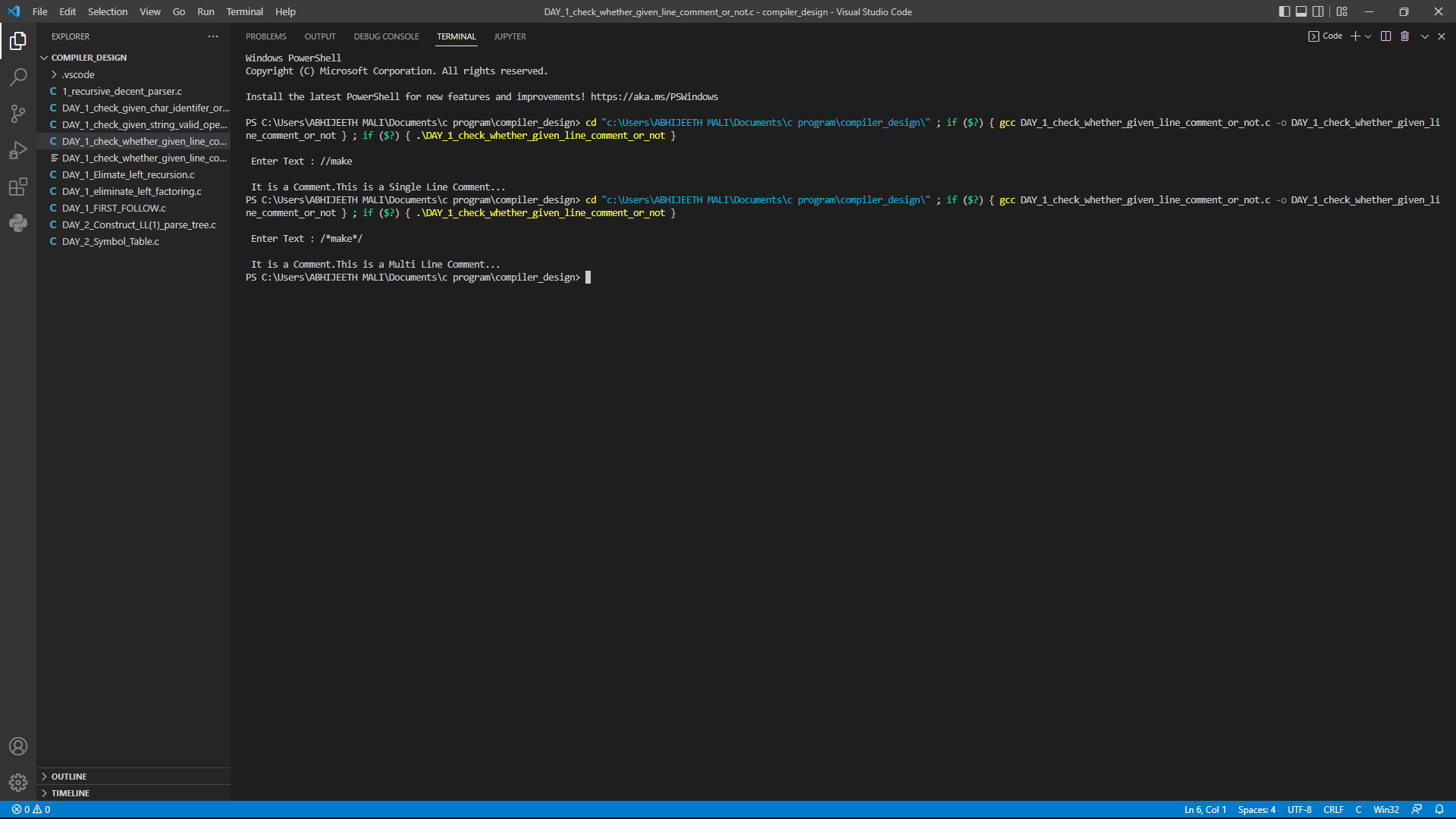


3. write a c program to find given line is comment or not

Source code :



Out put :



4. write a c program to implement left factoring

Source code :

#include <stdio.h>

#include <string.h>

int main()

{

    char gram[20], part1[20], part2[20], modifiedGram[20], newGram[20], tempGram[20];

    int i, j = 0, k = 0, l = 0, pos;

    printf("Enter Production : A->");

    gets(gram);

    for (i = 0; gram[i] != '|'; i++, j++)

        part1[j] = gram[i];

    part1[j] = '\0';

    for (j = ++i, i = 0; gram[j] != '\0'; j++, i++)

        part2[i] = gram[j];

    part2[i] = '\0';

    for (i = 0; i < strlen(part1) || i < strlen(part2); i++)

    {

        if (part1[i] == part2[i])

        {

            modifiedGram[k] = part1[i];

            k++;

            pos = i + 1;

        }

    }

    for (i = pos, j = 0; part1[i] != '\0'; i++, j++)

    {

        newGram[j] = part1[i];

    }

    newGram[j++] = '|';

    for (i = pos; part2[i] != '\0'; i++, j++)

    {

        newGram[j] = part2[i];

    }

    modifiedGram[k] = 'X';

    modifiedGram[++k] = '\0';

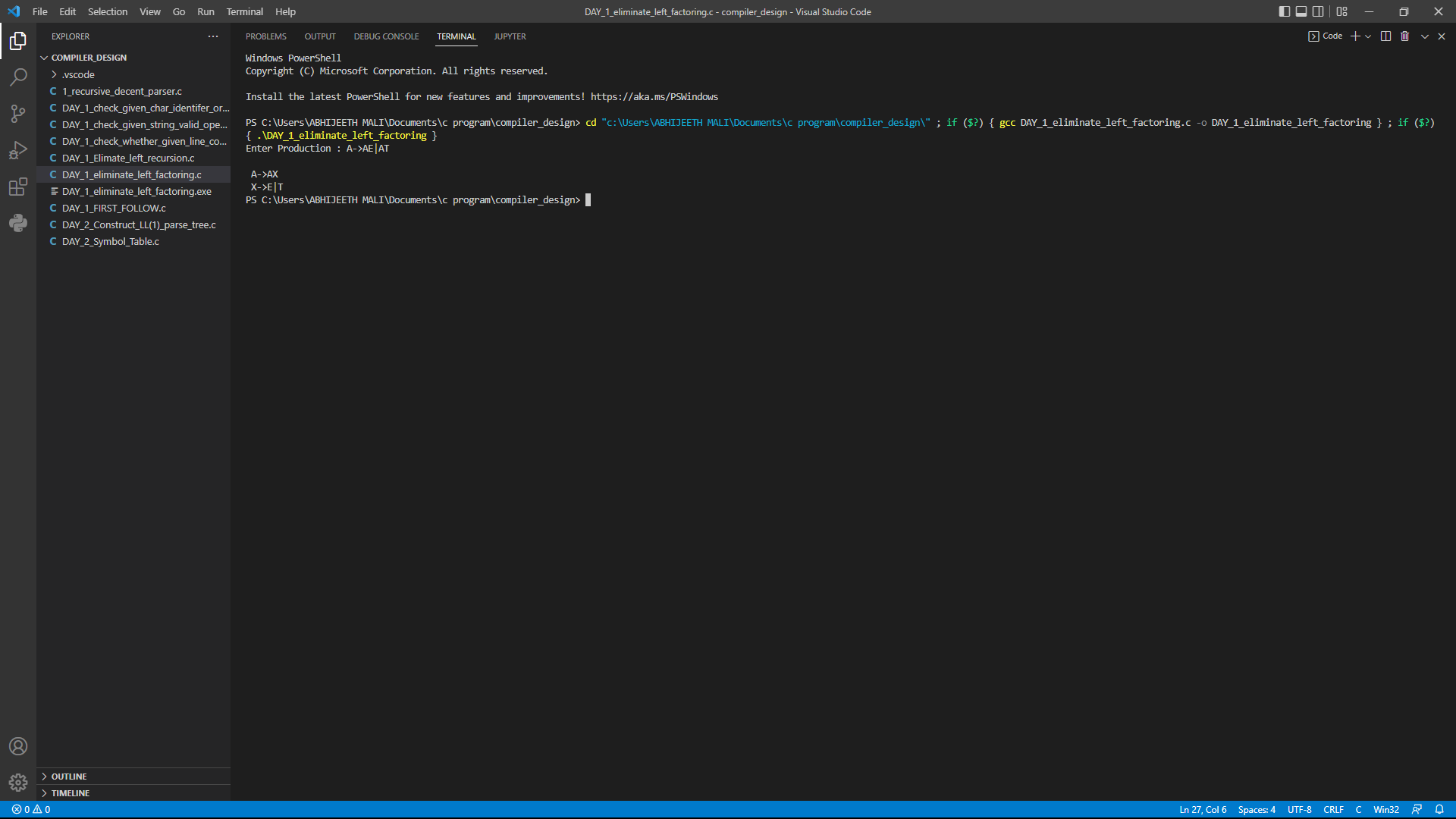
    newGram[j] = '\0';

    printf("\n A->%s", modifiedGram);

    printf("\n X->%s\n", newGram);

}

Output :



5.write a c program to implement left recursion

Source code :

#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

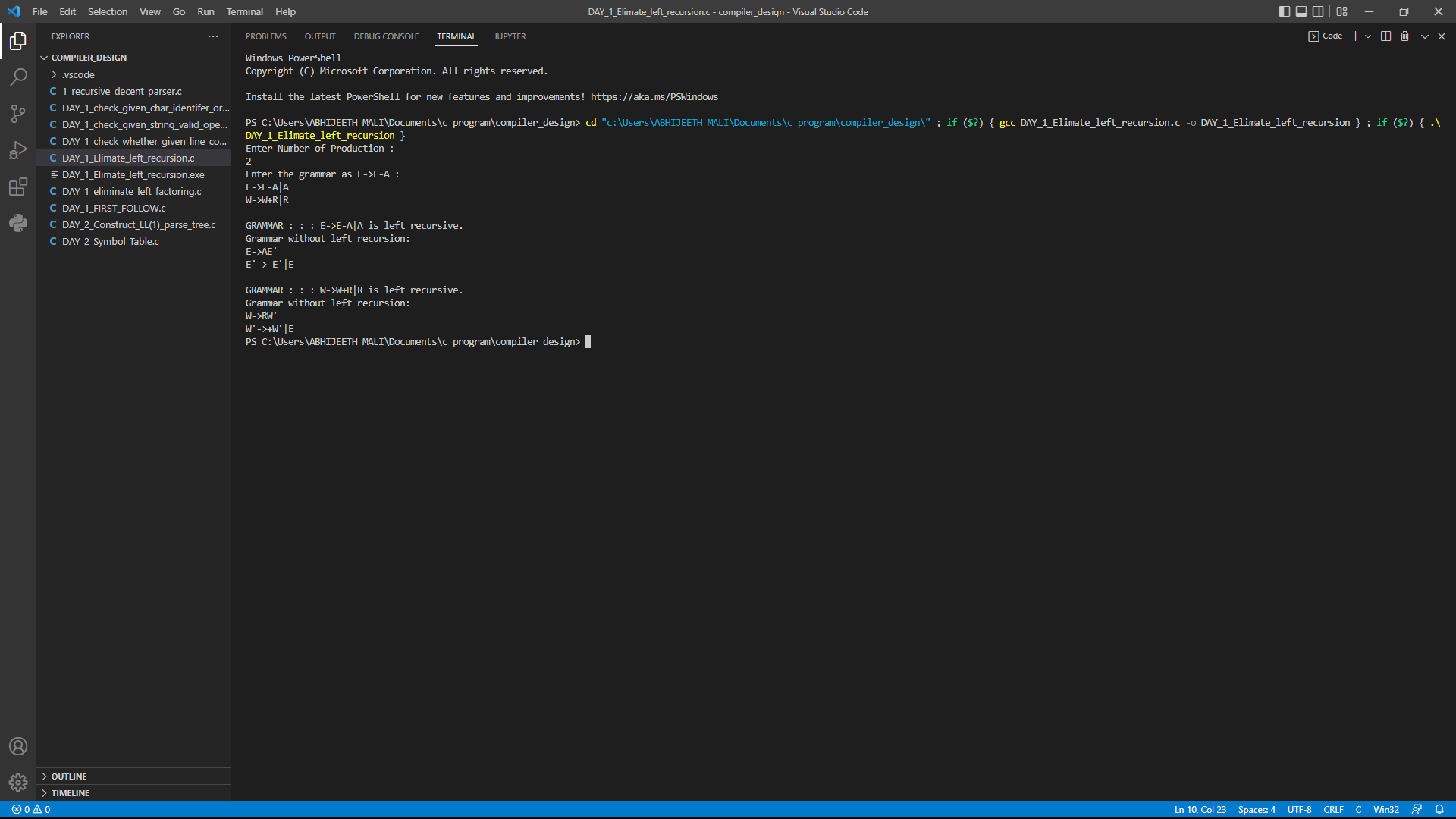
            printf(" is not left recursive.\n");

        index = 3;

    }

}

Output :



6.wrtie a c program to implement to do FIRST and FOLLOW.

Source code :

#include<stdio.h>

#include<ctype.h>

#include<string.h>

// Functions to calculate Follow

void followfirst(char, int, int);

void follow(char c);

// Function to calculate First

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;

    // The Input grammar

    strcpy(production[0], "E=TR");

    strcpy(production[1], "R=+TR");

    strcpy(production[2], "R=#");

    strcpy(production[3], "T=FY");

    strcpy(production[4], "Y=\*FY");

    strcpy(production[5], "Y=#");

    strcpy(production[6], "F=(E)");

    strcpy(production[7], "F=i");

    int kay;

    char done[count];

    int ptr = -1;

    for(k = 0; k < count; k++) {

        for(kay = 0; kay < 100; kay++) {

            calc\_first[k][kay] = '!';

        }

    }

    int point1 = 0, point2, xxx;

    for(k = 0; k < count; k++)

    {

        c = production[k][0];

        point2 = 0;

        xxx = 0;

        for(kay = 0; kay <= ptr; kay++)

            if(c == done[kay])

                xxx = 1;

        if (xxx == 1)

            continue;

        findfirst(c, 0, 0);

        ptr += 1;

        done[ptr] = c;

        printf("\n First(%c) = { ", c);

        calc\_first[point1][point2++] = c;

        for(i = 0 + jm; i < n; i++) {

            int lark = 0, chk = 0;

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

                if (first[i] == calc\_first[point1][lark])

                {

                    chk = 1;

                    break;

                }

            }

            if(chk == 0)

            {

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

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

            }

        }

        printf("}\n");

        jm = n;

        point1++;

    }

    printf("\n");

    printf("-----------------------------------------------\n\n");

    char donee[count];

    ptr = -1;

    for(k = 0; k < count; k++) {

        for(kay = 0; kay < 100; kay++) {

            calc\_follow[k][kay] = '!';

        }

    }

    point1 = 0;

    int land = 0;

    for(e = 0; e < count; e++)

    {

        ck = production[e][0];

        point2 = 0;

        xxx = 0;

        for(kay = 0; kay <= ptr; kay++)

            if(ck == donee[kay])

                xxx = 1;

        if (xxx == 1)

            continue;

        land += 1;

        follow(ck);

        ptr += 1;

        donee[ptr] = ck;

        printf(" Follow(%c) = { ", ck);

        calc\_follow[point1][point2++] = ck;

        for(i = 0 + km; i < m; i++) {

            int lark = 0, chk = 0;

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

            {

                if (f[i] == calc\_follow[point1][lark])

                {

                    chk = 1;

                    break;

                }

            }

            if(chk == 0)

            {

                printf("%c, ", f[i]);

                calc\_follow[point1][point2++] = f[i];

            }

        }

        printf(" }\n\n");

        km = m;

        point1++;

    }

}

void follow(char c)

{

    int i, j;

    if(production[0][0] == c) {

        f[m++] = '$';

    }

    for(i = 0; i < 10; i++)

    {

        for(j = 2;j < 10; j++)

        {

            if(production[i][j] == c)

            {

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

                {

                    followfirst(production[i][j+1], i, (j+2));

                }

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

                {

                    follow(production[i][0]);

                }

            }

        }

    }

}

void findfirst(char c, int q1, int q2)

{

    int j;

    if(!(isupper(c))) {

        first[n++] = c;

    }

    for(j = 0; j < count; j++)

    {

        if(production[j][0] == c)

        {

            if(production[j][2] == '#')

            {

                if(production[q1][q2] == '\0')

                    first[n++] = '#';

                else if(production[q1][q2] != '\0'

                        && (q1 != 0 || q2 != 0))

                {

                    findfirst(production[q1][q2], q1, (q2+1));

                }

                else

                    first[n++] = '#';

            }

            else if(!isupper(production[j][2]))

            {

                first[n++] = production[j][2];

            }

            else

            {

                findfirst(production[j][2], j, 3);

            }

        }

    }

}

void followfirst(char c, int c1, int c2)

{

    int k;

    if(!(isupper(c)))

        f[m++] = c;

    else

    {

        int i = 0, j = 1;

        for(i = 0; i < count; i++)

        {

            if(calc\_first[i][0] == c)

                break;

        }

        while(calc\_first[i][j] != '!')

        {

            if(calc\_first[i][j] != '#')

            {

                f[m++] = calc\_first[i][j];

            }

            else

            {

                if(production[c1][c2] == '\0')

                {

                    follow(production[c1][0]);

                }

                else

                {

                    followfirst(production[c1][c2], c1, c2+1);

                }

            }

            j++;

        }

    }

}

Output:

