Program Code

```
#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;

void main()
{
    int jm = 0;
    int km = 0;
    int i, choice;
    char c, ch;

    printf("\nEnter the number of rules: ");
    scanf("%d",&count);

    printf("\nEnter the productions : [A->B]");

    for(int i=0;i<count;i++)
    {
        scanf(" %s",production[i]);
    }

    char done[count];
    int ptr = -1;

    for(k = 0; k < count; k++)
    {
        for(int z = 0; z < 100; z++)
        {
            calc_first[k][z] = '!';
        }
    }
    int point1 = 0, point2, x;

    printf("\nFIRST");
    printf("\n=====");
    for(k = 0; k < count; k++)
    {
        c = production[k][0];
        point2 = 0;
        x = 0;
```

```
        for(int z = 0; z <= ptr; z++)
        {
            if(c == done[z])
                x = 1;
        }

        if (x == 1)
            continue;

        //Finding the First of the terminal of current production
        findfirst(c, 0, 0);

        done[++ptr] = c;
        printf("\n First(%c) = { ", c);
        calc_first[point1][point2++] = c;

        for(i = 0 + jm; i < n; i++)
        {
            int chk = 0;
            for(int l = 0; l < point2; l++)
            {
                 //checking if there is any repetition
                if (first[i] == calc_first[point1][l])
                {
                    chk = 1;
                    break;
                }
            }
            if(chk == 0)
            {
                printf("%c,", first[i]);
                calc_first[point1][point2++] = first[i];
            }
        }
        printf("}");
        jm = n;
        point1++;
    }
    printf("\n----------------------------------------\n\
n");
    printf("\nFOLLOW");
    printf("\n=====");
    char donee[count];
    ptr = -1;

    for(k = 0; k < count; k++)
    {
        for(int z = 0; z < 100; z++)
        {
            calc_follow[k][z] = '!';
        }
    }
    point1 = 0;
    int land = 0;
    for(e = 0; e < count; e++)
    {
        ck = production[e][0];
        point2 = 0;
        x = 0;
```

```
        for(int z = 0; z <= ptr; z++)
            if(ck == donee[z])
                x = 1;

        if (x == 1)
            continue;
        land += 1;

        follow(ck);

        donee[++ptr] = ck;
        printf("\nFollow(%c) = { ", ck);
        calc_follow[point1][point2++] = ck;

        for(i = 0 + km; i < m; i++)
        {
            int l = 0, chk = 0;
            for(l = 0; l < point2; l++)
            {
                if (f[i] == calc_follow[point1][l])
                {
                    chk = 1;
                    break;
                }
            }
            if(chk == 0)
            {
                printf("%c,", f[i]);
                calc_follow[point1][point2++] = f[i];
            }
        }
        printf(" }");
        km = m;
        point1++;
    }
    printf("\n");
}

void follow(char c)
{
    int i, j;

    if(production[0][0] == c)
    {
        f[m++] = '$';
    }
    for(i = 0; i < 10; i++)
    {
        for(j = 3;j < 10; j++)
        {
            if(production[i][j] == c)
            {
                if(production[i][j+1] != '\0')
                {
                    //if not the end of the production call
followfirst()
                    followfirst(production[i][j+1], i, (j+2));
                }
```

```
                if(production[i][j+1]=='\0' && c!=production[i]
[0])
                {
                    //else call follow() of the current non-
terminal of the production
                    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)
        {
            //checking if the first of 'c' is epsilon
            if(production[j][3] == '#')
            {
                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][3]))
            {
                //if terminal add terminal as first
                first[n++] = production[j][3];
            }
            else
            {
                //if non-terminal add call findfirst
                findfirst(production[j][3], j, 4);
            }
        }
    }
}

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] != '#')
            {
                //if the first of the current non_terminal is not
epsiln then add it to f
                f[m++] = calc_first[i][j];
            }
            else
            {
                if(production[c1][c2] == '\0')
                {
                    //if the end of the production is reached call
follow() of the non_terminal of the current production
                    follow(production[c1][0]);
                }
                else
                {
                    //else call followfirst() of the current
productions next symbol
                    followfirst(production[c1][c2], c1, c2+1);
                }
            }
            j++;
        }
    }
}
```

Output

```
students@pgcse-HP-280-G1-MT:~/Desktop/R7_66/R7_66/3/6$ ./first_follow

Enter the number of rules: 8

Enter the productions : [A->B]
E->TR
R->+TR
R->#
T->FY
Y->*FY
Y->#
F->(E)
F->i

FIRST
=====
 First(E) = { (,i,}
 First(R) = { +,#,}
 First(T) = { (,i,}
 First(Y) = { *,#,}
 First(F) = { (,i,}
-----------------------------------------------


FOLLOW
=====
Follow(E) = { $,), }
Follow(R) = { $,), }
Follow(T) = { +,$,), }
Follow(Y) = { +,$,), }
Follow(F) = { *,+,$,), }
```