**Compiler Design**

**FIRST AND FOLLOW**

Sai Ram Pendyala

RA1911003010696

**AIM:**

To write a program to perform first and follow using any language.

**ALGORITHM:**

For computing the first:

1. If X is a terminal, then FIRST(X) = {X}

Example: F -> I | id

We can write it as FIRST(F) -> { ( , id )

2. If X is a non-terminal like E -> T then to get FIRSTI substitute T with other productions

until you get a terminal as the first symbol

3. If X -> ε then add ε to FIRST(X).

For computing the follow:

1. Always check the right side of the productions for a non-terminal, whose FOLLOW set is

being found

2. (a) If that non-terminal (S, A, B…) is followed by any terminal (a,b…,\*,+,(,)…) , then add

that terminal into the FOLLOW set.

(b) If that non-terminal is followed by any other non-terminal, then add FIRST of other

nonterminal into the FOLLOW set.

**Program:**

#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;

// Stores the final result

// of the First Sets

*char* calc\_first[10][100];

// Stores the final result

// of the Follow Sets

*char* calc\_follow[10][100];

*int* m = 0;

// Stores the production rules

*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;

        // Checking if First of c has

        // already been calculated

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

            if(c == done[kay])

                xxx = 1;

        if (xxx == 1)

            continue;

        // Function call

        findfirst(c, 0, 0);

        ptr += 1;

        // Adding c to the calculated list

        done[ptr] = c;

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

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

        // Printing the First Sets of the grammar

        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;

    // Initializing the calc\_follow array

    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;

        // Checking if Follow of ck

        // has alredy been calculated

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

            if(ck == donee[kay])

                xxx = 1;

        if (xxx == 1)

            continue;

        land += 1;

        // Function call

        follow(ck);

        ptr += 1;

        // Adding ck to the calculated list

        donee[ptr] = ck;

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

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

        // Printing the Follow Sets of the grammar

        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;

    // Adding "$" to the follow

    // set of the start symbol

    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;

    // The case where we

    // encounter a Terminal

    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);

                }

            }

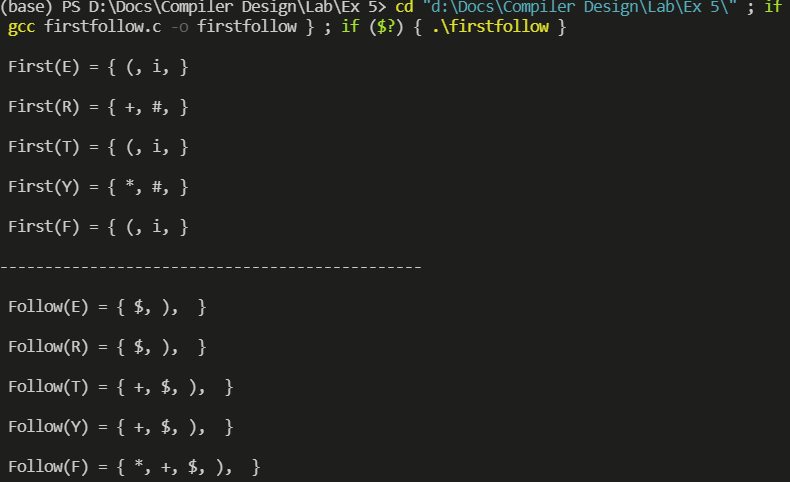
            j++;

        }

    }

}

**Output:**



**RESULT:**

The FIRST and FOLLOW sets of the non-terminals of a grammar were found

successfully using python language.