

FIRST AND FOLLOW

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

PROCEDURE:

FOR COMPUTING THE FIRST:

1. If X is a terminal then $\text{FIRST}(X) = \{X\}$
Example: $F \rightarrow I \mid id$
We can write it as $\text{FIRST}(F) \rightarrow \{ (, id)$
2. If X is a non-terminal like $E \rightarrow T$ then to get FIRST substitute T with other productions until you get a terminal as the first symbol.
3. If $X \rightarrow \epsilon$ then add ϵ to $\text{FIRST}(X)$.

FOR COMPUTING THE FOLLOW:

4. Always check the right side of the productions for a non-terminal, whose FOLLOW set is being found. (never see the left side).
5. If that non-terminal (S,A,B...) is followed by any terminal (a,b...,*,+,(),...) , then add that terminal into the FOLLOW set.
6. If that non-terminal is followed by any other non-terminal then add FIRST of other non-terminal into the FOLLOW set.

PROGRAM:

```
// C program to calculate the First and
// Follow sets of a given grammar
#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");

    printf("\nEnter the production 1:");
    scanf("%s",production[0]);

    printf("\nEnter the production 2:");
    scanf("%s",production[1]);

    printf("\nEnter the production 3:");
    scanf("%s",production[2]);

    printf("\nEnter the production 4:");
    scanf("%s",production[3]);

    printf("\nEnter the production 5:");
    scanf("%s",production[4]);

    printf("\nEnter the production 6:");
    scanf("%s",production[5]);

    printf("\nEnter the production 7:");

```

```

scanf("%s",production[6]);

printf("\nEnter the production 8:");
scanf("%s",production[7]);


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

// Initializing the calc_first array
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 already 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')
            {
                // Calculate the first of the next
                // Non-Terminal in the production
                followfirst(production[i][j + 1], i, (j + 2));
            }

            if (production[i][j + 1] == '\0' && c != production[i][0])
            {
                // Calculate the follow of the Non-Terminal
                // in the L.H.S. of the production
                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))
{
    // Recursion to calculate First of New
    // Non-Terminal we encounter after epsilon
    findfirst(production[q1][q2], q1, (q2 + 1));
}
else
    first[n++] = '#';
}
else if (!isupper(production[j][2]))
{
    first[n++] = production[j][2];
}
else
{
    // Recursion to calculate First of
    // New Non-Terminal we encounter
    // at the beginning
    findfirst(production[j][2], j, 3);
}
}
}
}
}

```

```

void followfirst(char c, int c1, int c2)
{
    int k;

    // The case where we encounter
    // a Terminal
    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;
        }

        // Including the First set of the
        // Non-Terminal in the Follow of
        // the original query
        while (calc_first[i][j] != '!')
        {

```

```

        if (calc_first[i][j] != '#')
        {
            f[m++] = calc_first[i][j];
        }
        else
        {
            if (production[c1][c2] == '\0')
            {
                // Case where we reach the
                // end of a production
                follow(production[c1][0]);
            }
            else
            {
                // Recursion to the next symbol
                // in case we encounter a "#"
                followfirst(production[c1][c2], c1, c2 + 1);
            }
        }
        j++;
    }
}

```

INPUT:

Enter the production 1: E=TR

Enter the production 2:R=+TR

Enter the production 3:R=#

Enter the production 4:T=FY

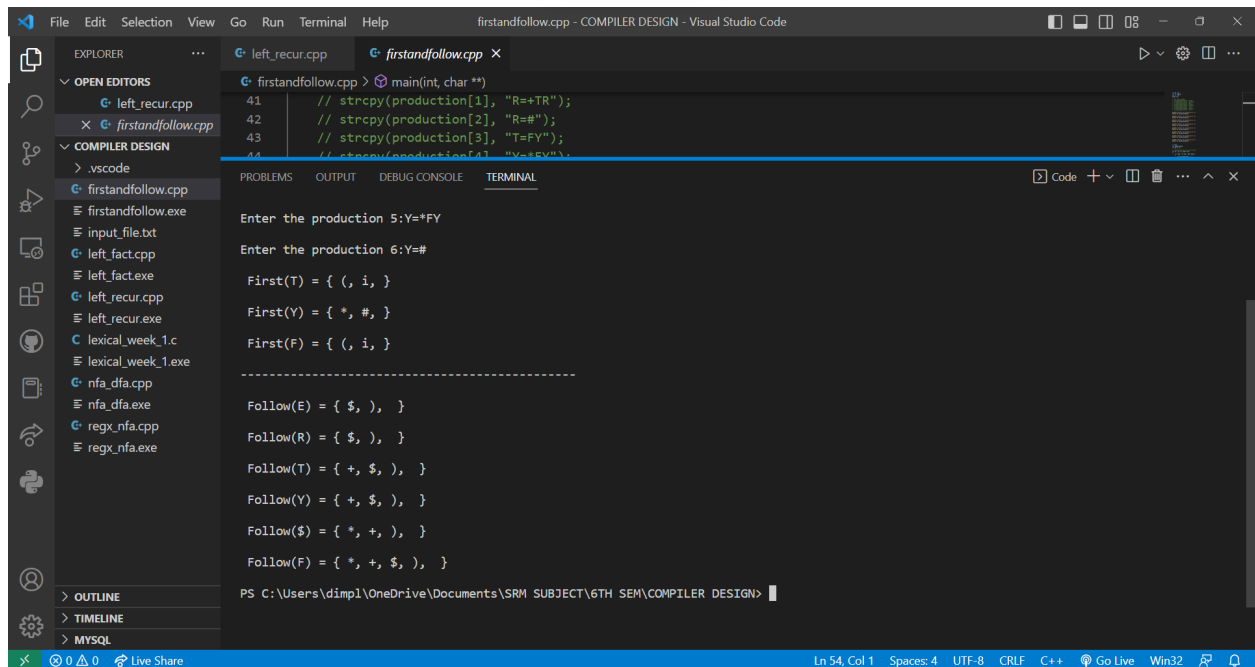
Enter the production 5:Y=*FY

Enter the production 6:Y=#

Enter the production 7:F=(E)

Enter the production 8:F=i

OUTPUT:



The screenshot shows the Visual Studio Code interface with the file `firststandfollow.cpp` open. The code defines a `main` function that takes an integer `char **` and uses `strcpy` to copy strings into an array. The output window shows the execution results for two test cases.

```
41 // strcpy(production[1], "R=+TR");
42 // strcpy(production[2], "R=#");
43 // strcpy(production[3], "T=FY");
44 // strcpy(production[4], "Y=SS");
```

Enter the production 5:Y=*FY
Enter the production 6:Y=#

First(T) = { (, i, }
First(Y) = { *, #, }
First(F) = { (, i, }

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

PS C:\Users\dimpl\OneDrive\Documents\SRM SUBJECT\6TH SEM\COMPILER DESIGN>

RESULT:

The FIRST and FOLLOW sets of non-terminals of a grammar were successfully found and implemented in C language.