ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

COMPILER DESIGN LAB

EXP 5 - FIRST AND FOLLOW ALGORITHM

AIM :

To create a C program and exhibit the implementation of first and follow algorithm on a given grammar

REQUIREMENTS :

1. Knowledge of the concepts of grammars and production rules
2. Knowledge of the concepts of First and Follow Algorithm
3. Online compiler for execution of C program

THEORY :

First and Follow Sets

An important part of parser table construction is to create first and follow sets. These sets can provide the actual position of any terminal in the derivation. This is done to

create the parsing table where the decision of replacing T[A, t] = α with some production rule.

First Set

This set is created to know what terminal symbol is derived in the first position by a non-terminal. For example,

$$\alpha \rightarrow t\,\beta$$

That is α derives t (terminal) in the very first position. So,

$t \in FIRST(\alpha)$.

$$FIRST(\alpha) = \{ t \mid \alpha \xrightarrow{*} t\,\beta \} \cup \{ \varepsilon \mid \alpha \xrightarrow{*} \varepsilon \}$$

Follow Set

Likewise, we calculate what terminal symbol immediately follows a non-terminal α in production rules. We do not consider what the non-terminal can generate but instead, we see what would be the next terminal symbol that follows the productions of a non-terminal.

Follow set can be seen as: FOLLOW(α) = { t | S *αt*}

ALGORITHM :

Algorithm for calculating First set

Look at the definition of FIRST(α) set:

- if α is a terminal, then FIRST(α) = { α }.

- if α is a non-terminal and α → ε is a production, then FIRST(α) = { ε }.
- if α is a non-terminal and α → □1 □2 □3 … □n and any FIRST(□) contains t then t is in FIRST(α).

First set can be seen as:

Algorithm for calculating Follow set:

- if α is a start symbol, then FOLLOW() = $
- if α is a non-terminal and has a production α → AB, then FIRST(B) is in FOLLOW(A) except ε.
- if α is a non-terminal and has a production α → AB, where B ε, then FOLLOW(A) is in FOLLOW(α).

SOURCE 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;

int main(int argc, char **argv)
{
    int jm = 0;
    int km = 0;
    int i, choice;
    char c, ch;
    count = 8;

    // The Input grammar
    printf("Enter production number 0");
    scanf("%s ",production[0]);
    printf("Enter production number 1");
    scanf("%s ",production[1]);
    printf("Enter production number 2");
    scanf("%s ",production[2]);
    printf("Enter production number 3");
    scanf("%s ",production[3]);
    printf("Enter production number 4");
    scanf("%s ",production[4]);
    printf("Enter production number 5");
    scanf("%s ",production[5]);
    printf("Enter production number 6");
    scanf("%s ",production[6]);
     printf("Enter production number 7");
    scanf("%s ",production[7]);

    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];
```

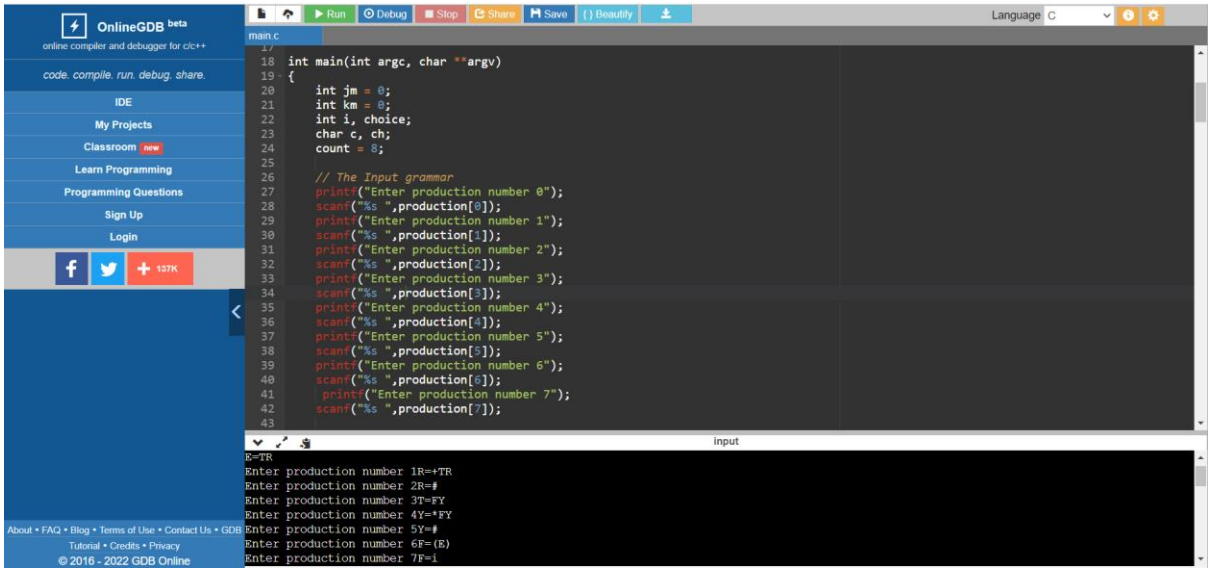
```
        point2 = 0;
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
            if(ck == donee[kay])
                xxx = 1;

        if (xxx == 1)
            continue;
        land += 1;
        follow(ck);
        ptr += 1;

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

        // 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;

    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++;
      }
    }
}
```

SCREENSHOT OF OUTPUT :

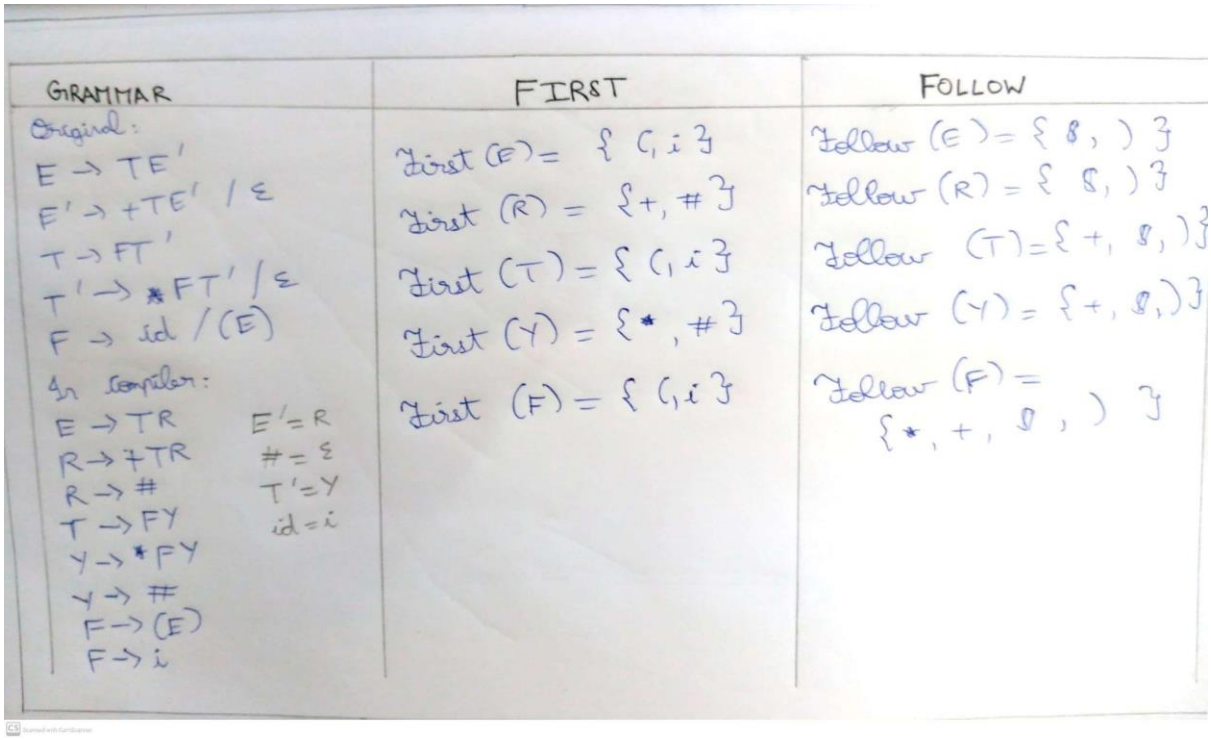



```
First(E) = { (, i, }

First(R) = { +, #, }

First(T) = { (, i, }

First(Y) = { *, #, }
```

```
First(F) = { (, i, }

------------------------------------------------

Follow(E) = { $, ),  }

Follow(R) = { $, ),  }
```

```
Follow(T) = { +, $, ),  }

Follow(Y) = { +, $, ),  }

Follow(F) = { *, +, $, ),  }
```

OBSERVATION :



GRAMMAR	FIRST	FOLLOW
Original: E → TE' E' → +TE' / ε T → FT' T' → *FT' / ε F → id / (E) In compiler: E → TR E'=R R → +TR #=ε R → # T'=Y T → FY id=i Y → *FY Y → # F → (E) F → i	First (E) = { (, i } First (R) = { +, # } First (T) = { (, i } First (Y) = { *, # } First (F) = { (, i }	Follow (E) = { $,) } Follow (R) = { $,) } Follow (T) = { +, $,) } Follow (Y) = { +, $,) } Follow (F) = { *, +, $,) }

The original grammar was analysed and the respective first and follow of all the symbols were found.

RESULT :

Thus we have successfully implemented the first and follow algorithm on a given input grammar.