Experiment 01 : First () and Follow() Set

Learning Objective: Student should be able to Compute First () and Follow () set of given grammar.

Tools: Jdk1.8,Turbo C/C++, Python, Notepad++

Theory:

1.Algorithmto Compute FIRST as follows:

- Let a be a string of terminals and non-terminals.
- First (a) is the set of all terminals that can begin strings derived from a.

Compute FIRST(X) as follows:

a) if X is a terminal, then FIRST(X)={X}

b) if X is a production, then add to FIRST(X)

c) if X is a non-terminal and $XY_1Y_2...Y_n$is a production, add FIRST(Y_i) to FIRST(X) if the preceding Y_js contain in their FIRSTs

2. Algorithm to Compute FOLLOW as follows:

a) FOLLOW(S) contains EOF

b) For productions AB, everything in FIRST () except goes into FOLLOW (B)

c) For productions AB or AB where FIRST () contains , FOLLOW(B) contains everything that is in FOLLOW(A)

Original grammar:

EE+E

EE*E

E(E)

Eid

This grammar is left-recursive, ambiguous and requires left-factoring. It needs to be modified before we build a predictive parser for it:

Step 1: Remove Ambiguity.

EE+T

TT*F

F(E)

Fid

Grammar is left recursive hence Remove left recursion:

ETE'

E'+TE'|

TFT'

T'*FT'|

F(E)

Fid

Step 2: Grammar is already left factored.

Step 3: Find First & Follow set to construct predictive parser table:-

FIRST (E) = FIRST(T) = FIRST(F) = {**(**, **id**}

FIRST (E') = {+, }
FIRST (T') = {*, }

FOLLOW (E) = FOLLOW(E') = {**$**, **)**}

FOLLOW (T) = FOLLOW(T') = {+, **$**, **)**}

FOLLOW (F) = {*, +, **$**, **)**}

Example:

ETE'

E'+TE'|

TFT'

T'*FT'|

F(E)

Fid

FIRST (E) = FIRST (T) = FIRST(F) = {(, **id**}

FIRST (E') = {+, }

FIRST (T') = {*, }

FOLLOW (E) = FOLLOW (E') = {**$**, **)**}

FOLLOW (T) = FOLLOW (T') = {+, **$**, **)**}

FOLLOW (F) = {*, +, **$**, **)**}

Application: To desige Top Down and Bottom up Parsers.

Design:

Result and Discussion:

Learning Outcomes: The student should have the ability to

LO1: Identify type of grammar G.
LO2: Define First () and Follow () sets.
LO3: ***Find*** First () and Follow () sets for given grammar G.
LO4: ***Apply*** First () and Follow () sets for designing Top Down and Bottom up Parsers

Course Outcomes: Upon completion of the course students will be able to analyze the analysis and synthesis phase of compiler for writhing application programs and construct different parsers for given context free grammars.

Conclusion:

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

CODE AND OUTPUT

```
#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;

// 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++;
            }
        }
}
```

OUTPUT:

```
C:\compb37\TE\SEM_6\spcc\firstfollow.exe

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

--------------------------------
Process exited after 1.001 seconds with return value 8
Press any key to continue . . .
```