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.Algorithm to 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→ε is a production, then add ε to FIRST(X)

c) if X is a non-terminal and $X \to Y_1 Y_2 ... Y_n$is a production, add FIRST(Y_i) to FIRST(X) if the preceding Y_js contain ε in their FIRSTs

2. Algorithm to Compute FOLLOW as follows:

a) FOLLOW(S) contains EOF

b) For productions A→αBβ, everything in FIRST (β) except ε goes into FOLLOW (B)

c) For productions A→αB or A→αBβ where FIRST (β) contains ε, FOLLOW(B)

contains everything that is in FOLLOW(A) Original grammar:

E→E+E
E→E*E
E→(E)
E→id

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.

E→E+T
T→T*F
F→(E)
F→id

Grammar is left recursive hence Remove left recursion:

E→TE'
E'→+TE'|ε
T→FT'
T'→*FT'|ε F→(E)
F→id

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') = {+, ε}
FIRST (T') = {*****, ε}
FOLLOW (E) = FOLLOW(E') = {**$**, **)**}
FOLLOW (T) = FOLLOW(T') = {+, **$**, **)**}
FOLLOW (F) = {*****, +, **$**, **)**}

Example:

E→TE'
E'→+TE'|ε
T→FT'
T'→*FT'|ε F→(E)
F→id
FIRST (E) = FIRST (T) = FIRST(F) = {**(**, **id**}
FIRST (E') = {+, ε}
FIRST (T') = {*, ε}
FOLLOW (E) = FOLLOW (E') = {$, **)**}
FOLLOW (T) = FOLLOW (T') = {+, $, **)**} FOLLOW
(F) = {*, +, $, **)**} **Application:**

To desige Top Down and Bottom up Parsers.

Implementation:

```
#include <ctype.h>
#include <stdio.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;
strcpy(production[0], "X=TnS");
strcpy(production[1], "X=Rm");
strcpy(production[2], "T=q");
strcpy(production[3], "T=#");
strcpy(production[4], "S=p");
strcpy(production[5], "S=#");
strcpy(production[6], "R=om");
strcpy(production[7], "R=ST");
      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");     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;
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] !=

followfirst(production[c1][c2], c1,
                                c2 +
1);
                }
'#') {                 f[m++]
= calc_first[i][j];
}             else {
if (production[c1][c2] ==
'\0') {
follow(production[c1][0]);
}                 else {

             }
 j++;
        }
    }
 }
```

OUTPUT:

```
First(X) = { q, n, o, p, #, }
 First(T) = { q, #, }
 First(S) = { p, #, }
 First(R) = { o, p, q, #, }

 Follow(X) = { $,  }
 Follow(T) = { n, m,  }
 Follow(S) = { $, q, m,  }
 Follow(R) = { m,  }
```

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 analyse 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				