# *Experiment No:06*

***Aim:*** To learn first and follow and write a program to find first(), follow() set of given grammar.

***Theory:***

**FIRST(X)** FOR ALL GRAMMAR SYMBOLS X

Apply following rules:

1. If **X** is terminal, **FIRST(X) = {X}**.
2. If X → ε is a production, then add ε to FIRST(X).
3. If X is a non-terminal, and X → Y1 Y2 … Yk is a production, and ε is in all of FIRST(Y1), …, FIRST(Yk), then add ε to FIRST(X).
4. If X is a non-terminal, and X → Y1 Y2 … Yk is a production, then add a to FIRST(X) if for some i, a is in FIRST(Yi), and ε is in all of FIRST(Y1), …, FIRST(Yi-1).

Applying rules 1 and 2 is obvious. Applying rules 3 and 4 for FIRST(Y1 Y2 … Yk) can be done as follows:

Add all the non-ε symbols of FIRST(Y1) to FIRST(Y1 Y2 … Yk). If ε ∈ FIRST(Y1), add all the non-ε symbols of FIRST(Y2). If ε ∈ FIRST(Y1) and ε ∈ FIRST(Y2), add all the non-ε symbols of FIRST(Y3), and so on. Finally, add ε to FIRST(Y1 Y2 … Yk) if ε ∈ FIRST(Yi), for all 1 ≤ i ≤ k.

***Example:***

Consider the following grammar.

E → E + T | T

T → T \* F | F

F → (E) | id

Grammar after removing left recursion:

E → TX

X → +TX | ε

T → FY

Y → \*FY | ε

F → (E) | id

For the above grammar, following the above rules, the FIRST sets could be computed as follows:

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

FIRST(X) = {+, ε}

FIRST(Y) = {\*, ε}

FOLLOW(A) FOR NON TERMINALS A

Apply the following rules:

1. If $ is the input end-marker, and S is the start symbol, $ ∈ FOLLOW(S).
2. If there is a production, A → αBβ, then (FIRST(β) – ε) ⊆ FOLLOW(B).
3. If there is a production, A → αB, or a production A → αBβ, where  ε ∈ FIRST(β), then FOLLOW(A) ⊆ FOLLOW(B).

***Note****that unlike the computation of* FIRST *sets for non-terminals, where the focus is on what a non-terminal generates, the computation of* FOLLOW *sets depends upon where the non-terminal appears on the RHS of a production.*

***Example:***

For the above grammar, the FOLLOW sets can be computed by applying the above rules as follows.

FOLLOW(E) = {$, )}

FOLLOW(E) ⊆ FOLLOW(X) [in other words, FOLLOW(X) contains FOLLOW(E)]

Since there is no other rule applicable to FOLLOW(X),

FOLLOW(X) = {$, )}

FOLLOW(T) ⊆ FOLLOW(Y)      …. (1)

(FIRST(X) – ε) ⊆ FOLLOW(T) i.e., {+} ⊆ FOLLOW(T)      …. (2)

Also, since ε ∈ FIRST(X), FOLLOW(E) ⊆ FOLLOW(T)

i.e., {$, )} ⊆ FOLLOW(T)     …. (3)

Putting (2) and (3) together, we get:

FOLLOW(T) = {$, ), +}

Since, there is no other rule applying to FOLLOW(Y), from (1), we get:

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

Since ε ∈ FIRST(Y), FOLLOW(T) ⊆ FOLLOW(F) and FOLLOW(Y) ⊆ FOLLOW(F). Also, (FIRST(Y) – ε) ⊆ FOLLOW(F). Putting all these together:

FOLLOW(F) = FOLLOW(T) ∪ FOLLOW(Y) ∪ (FIRST(Y) – ε) = {$, ), +, \*}

Here an example for you to follow through.

**The Grammar**

E → TE'

E' → +TE'

E' → ε

T → FT'

T' → \*FT'

T' → ε

F → (E)

F → id

|  |  |
| --- | --- |
| **First Sets** | **Follow Sets** |
| We Want to make First sets so first we list the sets we need  FIRST(E) = {}  FIRST(E') = {}  FIRST(T) = {}  FIRST(T') = {}  FIRST(F) = {}  First We apply rule 2 to T' → ε and E' → ε  FIRST(E) = {}  FIRST(E') = {ε}  FIRST(T) = {}  FIRST(T') = {ε}  FIRST(F) = {}  First We apply rule 3 to T' → \*FT' this rule tells us that we can add everything in FIRST(\*FT') into FIRST(T')  Since FIRST(\*) using rule 1 is \* we can add \* to FIRST(T')  FIRST(E) = {}  FIRST(E') = {+,ε}  FIRST(T) = {}  FIRST(T') = {\*,ε}  FIRST(F) = {}  First We apply rule 3 to T' → \*FT' this rule tells us that we can add everything in FIRST(\*FT') into FIRST(T')  Since FIRST(\*) using rule 1 is \* we can add \* to FIRST(T')  FIRST(E) = {}  FIRST(E') = {+,ε}  FIRST(T) = {}  FIRST(T') = {\*,ε}  FIRST(F) = {}  Two more productions begin with terminals F → (E) and F → id If we apply rule 3 to these we get...  FIRST(E) = {}  FIRST(E') = {+,ε}  FIRST(T) = {}  FIRST(T') = {\*,ε}  FIRST(F) = {'(',id}  Next we apply rule 3 to T → FT' once again this tells us that we can add FIRST(FT') to FIRST(T)  Since FIRST(F) doesn't contain ε that means that FIRST(FT') is just FIRST(F)  FIRST(E) = {}  FIRST(E') = {+,ε}  FIRST(T) = {'(',id}  FIRST(T') = {\*,ε}  FIRST(F) = {'(',id}  Lastly we apply rule 3 to E → TE' once again this tells us that we can add FIRST(TE') to FIRST(E)  Since FIRST(T) doesn't contain ε that means that FIRST(TE') is just FIRST(T)  FIRST(E) = {'(',id}  FIRST(E') = {+,ε}  FIRST(T) = {'(',id}  FIRST(T') = {\*,ε}  FIRST(F) = {'(',id}  Doing anything else doesn't change the sets so we are done! | We want to make Follow sets so first we list the sets we need  FOLLOW(E) = {}  FOLLOW(E') = {}  FOLLOW(T) ={}  FOLLOW(T') = {}  FOLLOW(F) = {}  The First thing we do is Add $ to the start Symbol 'E'  FOLLOW(E) = {$}  FOLLOW(E') = {}  FOLLOW(T) ={}  FOLLOW(T') = {}  FOLLOW(F) = {}  Next we apply rule 2 to E' → +TE' This says that everything in FIRST(E') except forε should be in FOLLOW(T)  FOLLOW(E) = {$}  FOLLOW(E') = {}  FOLLOW(T) ={+}  FOLLOW(T') = {}  FOLLOW(F) = {}  Next we apply rule 3 to E →TE' This says that we should add everything in FOLLOW(E) into FOLLOW(E')  FOLLOW(E) = {$}  FOLLOW(E') = {$}  FOLLOW(T) ={+}  FOLLOW(T') = {}  FOLLOW(F) = {}  Next we apply rule 3 to T → FT' This says that we should add everything in FOLLOW(T) into FOLLOW(T')  FOLLOW(E) = {$}  FOLLOW(E') = {$}  FOLLOW(T) ={+}  FOLLOW(T') = {+}  FOLLOW(F) = {}  Now we apply rule 2 to T' → \*FT' This says that everything in FIRST(T') except for ε should be in FOLLOW(F)  FOLLOW(E) = {$}  FOLLOW(E') = {$}  FOLLOW(T) ={+}  FOLLOW(T') = {+}  FOLLOW(F) = {\*}  Now we apply rule 2 to F → (E) This says that everything in FIRST(')') should be in FOLLOW(E)  FOLLOW(E) = {$,)}  FOLLOW(E') = {$}  FOLLOW(T) ={+}  FOLLOW(T') = {+}  FOLLOW(F) = {\*}  Next we apply rule 3 to E → TE' This says that we should add everything in FOLLOW(E) into FOLLOW(E')  FOLLOW(E) = {$,)}  FOLLOW(E') = {$,)}  FOLLOW(T) = {+}  FOLLOW(T') = {+}  FOLLOW(F) = {\*}  Next we apply rule 4 to E' → +TE' This says that we should add everything in FOLLOW(E') into FOLLOW(T) (because FIRST(E') contains ε)  FOLLOW(E) = {$,)}  FOLLOW(E') = {$,)}  FOLLOW(T) = {+,$,)}  FOLLOW(T') = {+}  FOLLOW(F) = {\*}  Next we apply rule 3 to T → FT' This says that we should add everything in FOLLOW(T) into FOLLOW(T')  FOLLOW(E) = {$,)}  FOLLOW(E') = {$,)}  FOLLOW(T) = {+,$,)}  FOLLOW(T') = {+,$,)}  FOLLOW(F) = {\*}  Finally we apply rule 4 to T' → \*FT' This says that we should add everything in FOLLOW(T') into FOLLOW(F)  FOLLOW(E) = {$,)}  FOLLOW(E') = {$,)}  FOLLOW(T) = {+,$,)}  FOLLOW(T') = {+,$,)}  FOLLOW(F) = {\*,+,$,)} |

***Program:***

#include<stdio.h>

#include<ctype.h>

void FIRST(char[],char );

void FOLLOW(char[], char );

void addToResultSet(char[],char);

int numOfProductions;

char productionSet[10][10];

void main()

{

int i, ch;

char choice;

char c;

char result[20], result1[20];

printf("How many number of productions ? :");

scanf(" %d",&numOfProductions);

for(i=0;i<numOfProductions;i++)//read production string eg: E=E+T

{

printf("Enter productions Number %d : ",i+1);

scanf(" %s",productionSet[i]);

}

do

{

printf("1. First or 2.Follow?");

scanf("%d", &ch);

if(ch==1)

{

printf("\n Find the FIRST of  :");

scanf(" %c",&c);

FIRST(result,c); //Compute FIRST; Get Answer in 'result' array

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

for(i=0;result[i]!='\0';i++)

printf(" %c ",result[i]);       //Display result

printf("}\n");

}

else if(ch==2)

{

printf("\n Find the FOLLOW of  :");

scanf(" %c",&c);

FOLLOW(result1,c); //Compute FIRST; Get Answer in 'result' array

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

for(i=0;result1[i]!='\0';i++)

printf(" %c ",result1[i]);       //Display result

printf("}\n");

}

printf("press 'y' to continue : ");

scanf(" %c",&choice);

}

while(choice=='y'||choice =='Y');

}

void FIRST(char\* Result,char c)

{

int i,j,k;

char subResult[20];

int foundEpsilon;

subResult[0]='\0';

Result[0]='\0';

//If X is terminal, FIRST(X) = {X}.

if(!(isupper(c)))

{

addToResultSet(Result,c);

return ;

}

//If X is non terminal

//Read each production

for(i=0;i<numOfProductions;i++)

{

//Find production with X as LHS

if(productionSet[i][0]==c)

{

//If X ? e is a production, then add e to FIRST(X).

if(productionSet[i][2]=='#') addToResultSet(Result,'#');

//If X is a non-terminal, and X ? Y1 Y2 … Yk

//is a production, then add a to FIRST(X)

//if for some i, a is in FIRST(Yi),

//and e is in all of FIRST(Y1), …, FIRST(Yi-1).

else

{

j=2;

while(productionSet[i][j]!='\0')

{

foundEpsilon=0;

FIRST(subResult,productionSet[i][j]);

for(k=0;subResult[k]!='\0';k++)

addToResultSet(Result,subResult[k]);

for(k=0;subResult[k]!='\0';k++)

if(subResult[k]=='#')

{

foundEpsilon=1;

break;

}

//No e found, no need to check next element

if(!foundEpsilon)

break;

j++;

}

}

}

}

return ;

}

void FOLLOW(char \*Result1, char c)

{

int i, j;

Result1[0]='\0';

char subresult[10], nchar;

subresult[0]='\0';

if(c==productionSet[0][0])

addToResultSet(Result1, '$');

else

{

for(i=0;i<numOfProductions;i++)

for(j=2; productionSet[i][j]!='\0'; j++)

{

if(productionSet[i][j]==c)

{

//printf("%c", productionSet[i][j]);

if(productionSet[i][j+1]=='\0')

{nchar=productionSet[i][0];

//printf("%c", nchar);

FOLLOW(Result1, nchar);

//printf("%s", Result1);

}

else

FIRST(Result1, productionSet[i][j+1]);

//if FIRST(B) contains epsilon

break;}

}

//printf("%s", Result1);}

}

void addToResultSet(char Result[],char val)

{

int k;

for(k=0 ;Result[k]!='\0';k++)

if(Result[k]==val)

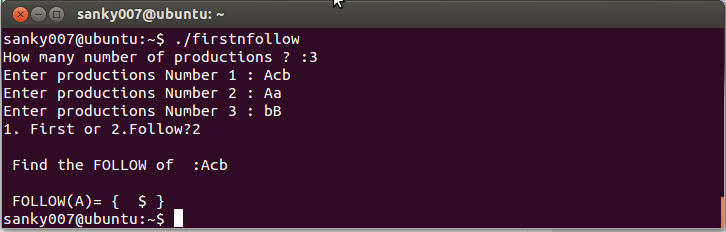
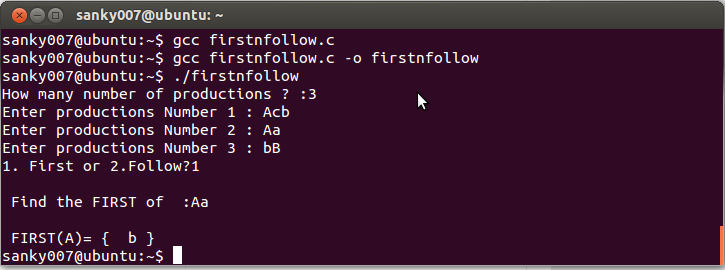
return;

Result[k]=val;

Result[k+1]='\0';

}

***Output:***



***Conclusion:*** Program for first(), follow() set of grammar successfully implemented.