**Experiment 7**

**Name:- Dishant Modh**

**Roll No:- IT076**

**Aim:** Write a program to calculate First and Follow set of the grammar.

**Input of program**- Grammar without any left factor and left recursion problem.

**Output of program-** First and Follow set of the Input grammar.

**Tools:** gcc compiler, Text editor.

**Code:**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

typedef struct production{

char lhs[3], rhs[10][10], first[20], follow[20], rfirst[10][20];

int length[10], subProduction;

}production;

production productions[20];

int noOfProduction;

void addMember(char ch, char \*first){

int j=0;

for( ; j<strlen(first); j++)

if(first[j]==ch)

return;

first[j++]=ch;

}

int findIndex(char ch){

for(int i=0; i<noOfProduction; i++)

if(productions[i].lhs[0] == ch)

return i;

return -1;

}

int haveNull(char \*s){

for(int i=0; i<strlen(s); i++)

if(s[i]=='^')

return 1;

return 0;

}

void copyData(char \*s1, char \*s2, int hasnull){

if(hasnull){

for(int m=0; m<strlen(s1); m++)

if(s1[m] != '^')

addMember(s1[m], s2);

}

else{

for(int m=0; m<strlen(s1); m++)

addMember(s1[m], s2);

}

}

void computeFirst(int i){

int j;

for(j=0; j<productions[i].subProduction; j++){

if(productions[i].rhs[j][0] == '^'){

addMember(productions[i].rhs[j][0], productions[i].first);

addMember(productions[i].rhs[j][0], productions[i].rfirst[j]);

}

else if(!isupper(productions[i].rhs[j][0])){

addMember(productions[i].rhs[j][0], productions[i].first);

addMember(productions[i].rhs[j][0], productions[i].rfirst[j]);

}

else{

int l = 0, k, m;

while(l < strlen(productions[i].rhs[j])){

if(!isupper(productions[i].rhs[j][l])){

addMember(productions[i].rhs[j][l],productions[i].first);

addMember(productions[i].rhs[j][l],productions[i].rfirst[j]);

break;

}

k = findIndex(productions[i].rhs[j][l]);

if(haveNull(productions[k].first)){

copyData(productions[k].first, productions[i].first, 1);

copyData(productions[k].first, productions[i].rfirst[j],1);

l++;

}

else{

copyData(productions[k].first, productions[i].first, 0);

copyData(productions[k].first, productions[i].rfirst[j],0);

break;

}

}

if(l == strlen(productions[i].rhs[j])){

addMember('^', productions[i].first);

addMember('^', productions[i].rfirst[j]);

}

}

}

}

void computeFollow(int i){

if(i == 0)

productions[i].follow[0]='$';

int j, k, l;

for(j=0; j<noOfProduction; j++){

for(k=0; k<productions[j].subProduction; k++){

for(l=0; l<strlen(productions[j].rhs[k]); l++){

if(productions[j].rhs[k][l]==productions[i].lhs[0]){

//printf("%c\n", productions[i].lhs[0]);

while(l<strlen(productions[j].rhs[k])){

if(productions[j].rhs[k][l+1] == 0 && productions[j].rhs[k][l] != productions[j].lhs[0]){

computeFollow(findIndex(productions[j].lhs[0]));

copyData(productions[j].follow,productions[i].follow, 0);

break;

}

else{

if(!isupper(productions[j].rhs[k][l+1])){

addMember(productions[j].rhs[k][l+1],productions[i].follow);

break;

}

int pos = findIndex(productions[j].rhs[k][l+1]);

if(haveNull(productions[pos].first)){

copyData(productions[pos].first,productions[i].follow, 1);

l++;

}

else{

copyData(productions[pos].first,productions[i].follow, 0);

break;

}

}

}

}

}

}

}

}

int main(){

int i, j;

printf("Enter total number of Productions : ");

scanf("%d", &noOfProduction);

for(i=0; i<noOfProduction; i++){

memset(productions[i].lhs, 0, sizeof(productions[i].lhs));

memset(productions[i].rhs, 0, sizeof(productions[i].rhs));

memset(productions[i].first, 0, sizeof(productions[i].first));

memset(productions[i].rfirst, 0, sizeof(productions[i].rfirst));

memset(productions[i].follow, 0, sizeof(productions[i].follow));

memset(productions[i].length, 0, sizeof(productions[i].length));

}

for(i=0; i<noOfProduction; i++){

printf("Enter LHS of production %d : ", i+1);

scanf("%s", productions[i].lhs);

printf("Enter total number of Subproductions of production %d : ",i+1);

scanf("%d", &productions[i].subProduction);

for(j=0; j<productions[i].subProduction; j++){

printf("Enter RHS of subproduction %d : ", j+1);

scanf("%s", productions[i].rhs[j]);

productions[i].length[j]=strlen(productions[i].rhs[j]);

}

}

printf("\nEntered Productions are : \n");

for(i=0; i<noOfProduction; i++){

printf("%s --> ", productions[i].lhs);

for(j=0; j<productions[i].subProduction; j++){

if(j!=0)

printf(" | %s", productions[i].rhs[j]);

else

printf("%s", productions[i].rhs[j]);

}

printf("\n");

}

for(i=(noOfProduction-1); i>(-1); i--)

computeFirst(i);

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

computeFollow(i);

printf("\nFIRST sets of Entered Productions are : \n");

for(i=0; i<noOfProduction; i++){

printf("FIRST(%s) : \n", productions[i].lhs);

for(int k=0; k<productions[i].subProduction; k++){

printf("\tFIRST(%s) : ", productions[i].rhs[k]);

for(int l=0; l<strlen(productions[i].rfirst[k]); l++){

if(l != 0)

printf(", %c", productions[i].rfirst[k][l]);

else

printf("{ %c", productions[i].rfirst[k][l]);

}

printf(" }\n");

}

for(j=0; j<strlen(productions[i].first); j++){

if(j != 0)

printf(", %c", productions[i].first[j]);

else

printf("\tFinal Answer : { %c", productions[i].first[j]);

}

printf(" }\n");

}

printf("\nFOLLOW sets of Entered Productions are : \n");

for(i=0; i<noOfProduction; i++){

printf("FOLLOW(%s) : ", productions[i].lhs);

for(j=0; j<strlen(productions[i].follow); j++){

if(j != 0)

printf(", %c", productions[i].follow[j]);

else

printf("{ %c", productions[i].follow[j]);

}

printf(" }\n");

}

return 0;

}

**Output:-**

