Lab program 5:

5. Write a program to compute FIRST set.

Aim: Program to compute FIRST set for a given grammar of non-terminals.

Algorithm:

The rules for finding FIRST of a given grammar is:

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 → Y_1 Y_2 … Y_k is a production, and ε is in all of FIRST(Y_1), …, FIRST(Y_k), then add ε to FIRST(X).
4. If X is a non-terminal, and X → Y_1 Y_2 … Y_k is a production, then add a to FIRST(X) if for some i, a is in FIRST(Y_i), and ε is in all of FIRST(Y_1), …, FIRST(Y_{i-1}).

Program:

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char);
void addToResultSet(char[],char);
int numOfProductions;
char productionSet[10][10];
main( )
{
    int i;
    char choice,c,result[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("\nFind the FIRST of:");
        scanf("%c",&c);
        FIRST(result,c);        //Compute FIRST; Get Answer in 'result' array
        printf("\nFIRST(%c)={",c);
        for(i=0;result[i]!='\0';i++)
            printf("%c",result[i]);          //Display result
            printf("}\n");
            printf("press 'y' to continue:");
            scanf("%c",&choice);
      }
     while(choice=='y'||choice=='Y');
}
/*
*Function FIRST:
```

```
*Compute the elements in FIRST(c) and write them in Result Array.
*/
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 nonterminal
    //Read each production
    for(i=0;i<numOfProductions;i++)
    {
        //Find production with X as LHS
        if(productionSet[i][0]==c)
        {
            //If X→ε is a production, then add ε to FIRST(X).
            if(productionSet[i][2]=='$')
                addToResultSet(Result,'$');
            //If X is a non-terminal, and X→Y1Y2…Yk is a production, then add a to FIRST(X)
            //if for some i, a is inFIRST(Yi),and ε 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;
                    }
                    //Noεfound,noneedtochecknextelement
                    if(!foundEpsilon)
                        break;
                    j++;
                }
            }
        }
    }
    return;
}
```

```
/*addToResultSet adds the computed element to result set.This code avoids multiple inclusion of elements */
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: