Practical No. 8

Aim: Write a program to find FIRST set of a given grammar.

Requirement: GCC

Theory:

FIRST set is defined for each non-terminal X in the grammar. It contains the first terminal symbol appearing at the body of each production for X. Following rule are applied to compute FIRST(X):

- If X is terminal, then FIRST (X) = {X}
- If X is a non-terminal such that, X → bc | a,

  FIRST (X) = {b, a}
- If X → BC

  B → b | ε

  C → c | ε

  FIRST (X) = {FIRST (B) + FIRST (C)} i.e. {b, c, ε}

For eg. E → T E’

E’ → + T E’ | ε

T → F T’

T’ → * F T’ | ε

F → (E) | id

FIRST (E) = {(, id}

FIRST (T) = {(, id}

FIRST (F) = {(, id}

FIRST (E’) = {+, ε}

FIRST (T’) = {*, ε}

Program:

```
#include <stdio.h>
#include <string.h>

#define MAX_NON_TERMINALS 10
#define MAX_TERMINALS 10
#define MAX_PRODUCTIONS 10

char nonTerminals[MAX_NON_TERMINALS];
char terminals[MAX_TERMINALS];
char productions[MAX_PRODUCTIONS][50];
int numProductions;

// Function to add a terminal to the FIRST set
void addToFirst(char firstSet[], char terminal)
{
        int i;
        for (i = 0; firstSet[i] != '\0'; i++)
        {
            if (firstSet[i] == terminal)
            {
                return; // Terminal already in FIRST set
            }
        }
        firstSet[i] = terminal;
        firstSet[i + 1] = '\0';
    }

    // Function to calculate the FIRST set for a given non-terminal
    void calculateFirst(char nonTerminal, char firstSet[])
    {
        int i, j;
        for (i = 0; i < numProductions; i++)
        {
            if (productions[i][0] == nonTerminal)
            {
                if (productions[i][3] == '|')
                {
                    // Production is of the form A -> a...
                    addToFirst(firstSet, productions[i][4]);
                }
                else
                {
                    // Production is of the form A -> a...
                    for (j = 3; productions[i][j] != '\0'; j++)
                    {
                        if (productions[i][j] == '|')
                        {
                            break;
                        }
                    if (strchr(terminals, productions[i][j]) != NULL)
                        {
                        addToFirst(firstSet, productions[i][j]);
                            break;
                        }
                    else  if  (strchr(nonTerminals,  productions[i][j])  !=
NULL)
                        {
                        calculateFirst(productions[i][j], firstSet);
                        if (strchr(firstSet, 'e') == NULL)
                            {
                                break;
                            }
                        }
                    }
                }
            }
        }
    }

    int main()
    {
        int i;

        printf("Enter non-terminals (without space in-between): ");
        gets(nonTerminals);

        printf("Enter terminals (without space in-between): ");
        gets(terminals);

        printf("Enter number of productions: ");
        scanf("%d", &numProductions);
        getchar();

        printf("Enter productions (A -> alpha format):\n");
        for (i = 0; i < numProductions; i++)
        {
            gets(productions[i]);
        }

        // Calculate the FIRST set for each non-terminal
        printf("FIRST Set:\n");
        for (i = 0; nonTerminals[i] != '\0'; i++)
        {
            char firstSet[MAX_TERMINALS] = {'\0'};
            calculateFirst(nonTerminals[i], firstSet);
            printf("FIRST(%c) = {%s}\n", nonTerminals[i], firstSet);
        }

        return 0;
    }
```

Output:

//Paste a color printout of the output here.

Conclusion:

Thus, a program to find the FIRST set of all the non-terminals in the given grammar is implemented successfully.