# Week 6 Assignment

Write a C program for the computation of FIRST and FOLLOW for a given CFG

## Program

```
#include <iostream>
#include <string>
#include <map>
#include <vector>
#include <set>
#include <algorithm>
using namespace std;

class Grammar
{
private:
    int noOfProductions;
    int noOfTerminals;
    int noOfNonTerminals;
    map<char, vector<string>> grammar;
    set<char> nonTerminals;
    set<char> terminals;
    map<char, set<char>> first;
    map<char, set<char>> follow;
    map<char, set<char>> oldFollow;

public:
    Grammar() {}
    Grammar(int noOfProuductions)
    {
        this->noOfProductions = noOfProductions;
    }
```

```
Grammar(int noOfTerminals, int noOfNonTerminals, int noOfProductions)
{
   this->noOfTerminals = noOfTerminals;
   this->noOfNonTerminals = noOfNonTerminals;
   this->noOfProductions = noOfProductions;
}
void setTerminals()
{
   char terminal;
   for (int i = 0; i < noOfTerminals; i++)
   {
      cin >> terminal;
      terminals.insert(terminal);
   }
}
void setNonTerminals()
{
   char nonTerminal;
   for (int i = 0; i < noOfNonTerminals; i++)
   {
      cin >> nonTerminal;
      nonTerminals.insert(nonTerminal);
   }
}
void setProductions()
{
   char nonTerminal;
   string production;
   for (int i = 0; i < noOfProductions; i++)
   {
```

```
            cout << "Enter the Left Handside(Non Terminal) of the " << i + 1 << " production: ";
            cin >> nonTerminal;
            cout << "Enter the Right Handisde(Production rule) of the " << i + 1 << " production:
";
            cin >> production;
            grammar[nonTerminal].push_back(production);
            nonTerminals.insert(nonTerminal);
        }
    }
    void setFirst()
    {
        set<char> firstSet;
        for (char nonTerminal : nonTerminals)
        {
            firstSet = calculateFirst(nonTerminal);
            first[nonTerminal].insert(firstSet.begin(), firstSet.end());
        }
    }
    set<char> calculateFirst(char ch)
    {
        set<char> firstSet;
        if (terminals.find(ch) != terminals.end())
        {
            firstSet.insert(ch);
        }
        else
        {
            for (string production : grammar[ch])
            {
                char firstSymbol = production[0];
                if (first.find(firstSymbol) != first.end())
```

```
            {
                firstSet.insert(first[firstSymbol].begin(), first[firstSymbol].end());
            }
            else
            {
                set<char> firstSymbolFirst = calculateFirst(firstSymbol);
                firstSet.insert(firstSymbolFirst.begin(), firstSymbolFirst.end());
            }
        }
    }
    return firstSet;
}
void setFollow()
{
    for (char nonTerminal : nonTerminals)
    {
        follow[nonTerminal] = {};
    }
    follow['E'].insert('$');
    bool changed = true;
    while (changed)
    {
        changed = false;

        for (char nonTerminal : nonTerminals)
        {
            for (auto it = grammar.begin(); it != grammar.end(); it++)
            {
                char leftHandSide = it->first;
                vector<string> productions = it->second;
```

```
for (string production : productions)
{
   for (int i = 0; i < production.length(); i++)
   {
      if (production[i] == nonTerminal)
      {
         for (int j = i + 1; j < production.length(); j++)
         {
            char symbol = production[j];
            if (terminals.find(symbol) != terminals.end())
            {
               follow[nonTerminal].insert(symbol);
               break;
            }
            else
            {
               set<char> firstBeta = calculateFirst(symbol);
               if (firstBeta.find('@') == firstBeta.end())
               {
                  follow[nonTerminal].insert(firstBeta.begin(), firstBeta.end());
                  break;
               }
               else
               {

                  firstBeta.erase('@');
                  follow[nonTerminal].insert(firstBeta.begin(), firstBeta.end());
                  if (j == production.length() - 1)
                  {
```

```
                                set<char> followA = follow[leftHandSide];
                                follow[nonTerminal].insert(followA.begin(), followA.end());
                            }
                        }
                    }
                }
                if (i == production.length() - 1)
                {
                    set<char> followA = follow[leftHandSide];
                    follow[nonTerminal].insert(followA.begin(), followA.end());
                }
            }
        }
    }
    for (char nonTerminal : nonTerminals)
    {
        if (follow[nonTerminal] != oldFollow[nonTerminal])
        {
            changed = true;
            oldFollow[nonTerminal] = follow[nonTerminal];
        }
    }
  }
}
map<char, set<char>> getFirst()
{
  return first;
}
```

```
    map<char, set<char>> getFollow()
    {
        return follow;
    }
};

int main()
{
    int noOfProductions, noOfTerminals, noOfNonTerminals;
    cout << "Enter the no of terminals: ";
    cin >> noOfTerminals;
    cout << "Enter the no of non terminals: ";
    cin >> noOfNonTerminals;
    cout << "Enter the no of productions: ";
    cin >> noOfProductions;
    Grammar g(noOfTerminals, noOfNonTerminals, noOfProductions);
    cout << "Enter the terminals" << endl;
    g.setTerminals();
    cout << "Enter the non terminals" << endl;
    g.setNonTerminals();
    cout << "Enter the Productions" << endl;
    g.setProductions();
    g.setFirst();
    g.setFollow();
    map<char, set<char>> first = g.getFirst();
    map<char, set<char>> follow = g.getFollow();
    for (auto pair : first)
    {
        cout << "First(" << pair.first << ")"
             << " = ";
```

```
        for (auto terminal : pair.second)
        {
            cout << terminal << "  ";
        }
        cout << endl;
    }
    for (auto pair : follow)
    {
        cout << "Follow(" << pair.first << ")"
             << " = ";
        for (auto terminal : pair.second)
        {
            cout << terminal << "  ";
        }
        cout << endl;
    }
    return 0;
}
```

## Input:

Enter the no of terminals: 6

Enter the no of non terminals: 5

Enter the no of productions: 8

Enter the terminals

+

*

(

)

i

@

Enter the non terminals

E

A

T

B

F

Enter the Productions

Enter the Left Handside(Non Terminal) of the 1 production: E

Enter the Right Handisde(Production rule) of the 1 production: TA

Enter the Left Handside(Non Terminal) of the 2 production: A

Enter the Right Handisde(Production rule) of the 2 production: +TA

Enter the Left Handside(Non Terminal) of the 3 production: A

Enter the Right Handisde(Production rule) of the 3 production: @

Enter the Left Handside(Non Terminal) of the 4 production: T

Enter the Right Handisde(Production rule) of the 4 production: FB

Enter the Left Handside(Non Terminal) of the 5 production: B

Enter the Right Handisde(Production rule) of the 5 production: *FB

Enter the Left Handside(Non Terminal) of the 6 production: B

Enter the Right Handisde(Production rule) of the 6 production: @

Enter the Left Handside(Non Terminal) of the 7 production: F

Enter the Right Handisde(Production rule) of the 7 production: (E)

Enter the Left Handside(Non Terminal) of the 8 production: F

Enter the Right Handisde(Production rule) of the 8 production: @

## Output:

First(A) = + @

First(B) = * @

First(E) = ( @

First(F) = ( @

First(T) = ( @

Follow(A) = $ )

Follow(B) = $ ) +

Follow(E) = $ )

Follow(F) = $ ) * +

Follow(T) = $ ) +