Experiment – 5

AIM- Write a program to find first of given production of grammar.

Description -

FIRST

FIRST(X) for a grammar symbol X is the set of terminals that begin the strings derivable from X.

Rules to compute FIRST set:

1. If x is a terminal, then FIRST(x) = { ‘x’ }

2. If x-> Є, is a production rule, then add Є to FIRST(x).

3. If X->Y1 Y2 Y3....Yn is a production,

1. FIRST(X) = FIRST(Y1)

2. If FIRST(Y1) contains Є then FIRST(X) = { FIRST(Y1) – Є } U { FIRST(Y2) }

3. If FIRST (Yi) contains Є for all i = 1 to n, then add Є to FIRST(X).

Code -

```
#include <bits/stdc++.h>
using namespace std;

map<string, vector<string> > grammar;

bool isCapital(char ch){
    if(ch>='A' && ch<='Z'){
        return true;
    }
    return false;
}
bool ifEpsilon(set<string> s){
    return (s.count("^") > 0) ? true : false;
}
void setPrint(set<string> s){
    for(auto i:s){
```

```
      cout<<i<<", ";
   }
}
void setUnion(set<string> &s1, set<string> &s2){
   for(auto i:s2){
      if(i != "^"){
         s1.insert(i);
      }
   }
   return;
}
void calcFirst(string nonTerminal, set<string> &firstTemp){
   vector<string> prods = grammar[nonTerminal];
   bool epsilon = false;

   for(auto p:prods){
      set<string> temp;
      string prod = p;

      if(prod=="^"){
         firstTemp.insert("^");
         continue;
      }
      if(!isCapital(prod[0])){
         firstTemp.insert(string(1, prod[0]));
         continue;
      }
      calcFirst(string(1, prod[0]), temp);

      setUnion(firstTemp, temp);

      if(ifEpsilon(temp)){
         int j = 1;

         while(j<prod.size() && ifEpsilon(temp) && isCapital(prod[j])){
            temp.clear();
            calcFirst(string(1, prod[j]), temp);
            setUnion(firstTemp, temp);
            j++;
         }
         if(j==prod.size() && ifEpsilon(temp)){
            epsilon = true;
```

```
            }
            if(j<prod.size() && ifEpsilon(temp)) {
                firstTemp.insert(string(1, prod[j]));
            }
        }
    }
    if(epsilon){
        firstTemp.insert("^");
    }
    return;
}

int main(int argc, char const *argv[]){
    int nProds;

    cout<<"Enter the no. of non-terminals: ";
    cin>>nProds;

    for (int i=0;i<nProds;i++){
        cout<<"\nEnter the non-terminal: ";
        string str;
        cin>>str;

        grammar[str] = vector<string> ();
        cout<<"Enter the number of productions: ";
        int n;
        cin>>n;

        cout<<"Enter the productions from '"
                <<str
                <<"' (space separated): ";
        for (int j=0;j<n;j++){
            string temp;
            cin>>temp;
            grammar[str].push_back(temp);
        }
    }
    cout<<"\nEnter start symbol: ";
    string startSymbol;
    cin>>startSymbol;

    cout<<"\nFirst of Non-Terminals in Given Grammer: \n";
```

```
    for(auto p:grammar){
        cout<<"\t";
        set<string> firstTemp;
        calcFirst(p.first, firstTemp);
        cout<<p.first<<" => { ";
        setPrint(firstTemp);
        cout<<"}"<<endl;
    }
    return 0;
}
```

Output –

```
File  Edit  View  Search  Terminal  Help
prince@pp-asus:~/lab/CD_lab/5.First$ g++ code.cpp
prince@pp-asus:~/lab/CD_lab/5.First$ ./a.out
Enter the no. of non-terminals: 5

Enter the non-terminal: E
Enter the number of productions: 1
Enter the productions from 'E' (space separated): TR

Enter the non-terminal: R
Enter the number of productions: 2
Enter the productions from 'R' (space separated): +TR ^

Enter the non-terminal: T
Enter the number of productions: 1
Enter the productions from 'T' (space separated): FY

Enter the non-terminal: Y
Enter the number of productions: 2
Enter the productions from 'Y' (space separated): *FY ^

Enter the non-terminal: F
Enter the number of productions: 2
Enter the productions from 'F' (space separated): n (E)

Enter start symbol: E

First of Non-Terminals in Given Grammer:
        E => { (, n, }
        F => { (, n, }
        R => { +, ^, }
        T => { (, n, }
        Y => { *, ^, }
prince@pp-asus:~/lab/CD_lab/5.First$
```

Learnings – First and follow helps in the implementation of many parsers. It helps the parsers to apply the proper needed rule at the correct position.