Name – Bhargav Shamuvel Gurav | **PRN** – 2041009

Class – L.Y. B-Tech (Computer) | **Batch** – B1

Course Code – CO406U | **Course Name** - CDL

Practical no. 5

Aim: Simulate First and Follow of a Grammar.

Theory :
FIRST and FOLLOW are two functions associated with grammar that help us fill in the entries of an M-table.

FIRST ()− It is a function that gives the set of terminals that begin the strings derived from the production rule.

A symbol c is in FIRST (α) if and only if $\alpha \Rightarrow c\beta$ for some sequence β of grammar symbols.

A terminal symbol a is in FOLLOW (N) if and only if there is a derivation from the start symbol S of the grammar such that $S \Rightarrow \alpha N\alpha\beta$, where α and β are a (possible empty) sequence of grammar symbols. In other words, a terminal c is in FOLLOW (N) if c can follow N at some point in a derivation.

Benefit of FIRST ( ) and FOLLOW ( )

- It can be used to prove the LL (K) characteristic of grammar.
- It can be used to promote in the construction of predictive parsing tables.
- It provides selection information for recursive descent parsers.

Computation of FIRST
FIRST (α) is defined as the collection of terminal symbols which are the first letters of strings derived from α.

FIRST $(\alpha) = \{\alpha \mid \alpha \rightarrow * \alpha\beta$ for some string $\beta\}$

If X is Grammar Symbol, then First (X) will be −

If X is a terminal symbol, then FIRST(X) = {X}
If $X \rightarrow \varepsilon$, then FIRST(X) = {ε}
If X is non-terminal & $X \rightarrow a\ \alpha$, then FIRST (X) = {a}

If X → Y1, Y2, Y3, then FIRST (X) will be
(a) If Y is terminal, then

FIRST (X) = FIRST (Y1, Y2, Y3) = {Y1}

(b) If Y1 is Non-terminal and

If Y1 does not derive to an empty string i.e., If FIRST (Y1) does not contain ε then, FIRST (X) = FIRST (Y1, Y2, Y3) = FIRST(Y1)

(c) If FIRST (Y1) contains ε, then.

FIRST (X) = FIRST (Y1, Y2, Y3) = FIRST(Y1) − {ε} ∪ FIRST(Y2, Y3)

Similarly, FIRST (Y2, Y3) = {Y2}, If Y2 is terminal otherwise if Y2 is Non-terminal then

FIRST (Y2, Y3) = FIRST (Y2), if FIRST (Y2) does not contain ε.

If FIRST (Y2) contain ε, then

FIRST (Y2, Y3) = FIRST (Y2) − {ε} ∪ FIRST (Y3)

Similarly, this method will be repeated for further Grammar symbols, i.e., for Y4, Y5, Y6 … . YK.

Computation of FOLLOW

Follow (A) is defined as the collection of terminal symbols that occur directly to the right of A.

FOLLOW(A) = {a|S ⇒* αAaβ where α, β can be any strings}

Rules to find FOLLOW

If S is the start symbol, FOLLOW (S) ={$}

If production is of form A → α B β, β ≠ ε.

(a) If FIRST (β) does not contain ε then, FOLLOW (B) = {FIRST (β)}

Or

(b) If FIRST (β) contains ε (i. e. , β ⇒* ε), then

FOLLOW (B) = FIRST (β) − {ε} ∪ FOLLOW (A)

∵ when β derives ε, then terminal after A will follow B.

If production is of form A → αB, then Follow (B) ={FOLLOW (A)}.

Program Code:

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

set<char> ss;
bool dfs(char i, char org, char last, map<char,vector<vector<char>>> &mp){
   bool rtake = false;
   for(auto r : mp[i]){
      bool take = true;
      for(auto s : r){
         if(s == i) break;
         if(!take) break;
         if(!(s>='A'&&s<='Z')&&s!='e'){
            ss.insert(s);
            break;
         }
         else if(s == 'e'){
            if(org == i||i == last)
            ss.insert(s);
            rtake = true;
            break;
         }
         else{
            take = dfs(s,org,r[r.size()-1],mp);
            rtake |= take;
         }
      }
   }
   return rtake;
```

```
}

int main(){
    int i,j;
    ifstream fin("inputfirstfollow.txt");
    string num;
    vector<int> fs;
    vector<vector<int>> a;
    map<char,vector<vector<char>>> mp;
    char start;
    bool flag = 0;
    cout<<"Grammar: "<<'\n';
    while(getline(fin,num)){
        if(flag == 0) start = num[0],flag = 1;
        cout<<num<<'\n';
        vector<char> temp;
        char s = num[0];
        for(i=3;i<num.size();i++){
            if(num[i] == '|'){
                mp[s].push_back(temp);
                temp.clear();
            }
            else temp.push_back(num[i]);
        }
        mp[s].push_back(temp);
    }
    map<char,set<char>> fmp;
    for(auto q : mp){
        ss.clear();
        dfs(q.first,q.first,q.first,mp);
        for(auto g : ss) fmp[q.first].insert(g);
    }

    cout<<'\n';
    cout<<"FIRST: "<<'\n';
    for(auto q : fmp){
        string ans = "";
        ans += q.first;
        ans += " = {";
        for(char r : q.second){
```

```
        ans += r;
        ans += ',';
    }
    ans.pop_back();
    ans+="}";
    cout<<ans<<'\n';
}

map<char,set<char>> gmp;
gmp[start].insert('$');
int count = 10;
while(count--){
    for(auto q : mp){
        for(auto r : q.second){
            for(i=0;i<r.size()-1;i++){
                if(r[i]>='A'&&r[i]<='Z'){
                    if(!(r[i+1]>='A'&&r[i+1]<='Z')) gmp[r[i]].insert(r[i+1]);
                    else {
                        char temp = r[i+1];
                        int j = i+1;
                        while(temp>='A'&&temp<='Z'){
                            if(*fmp[temp].begin()=='e'){
                                for(auto g : fmp[temp]){
                                    if(g=='e') continue;
                                    gmp[r[i]].insert(g);
                                }
                                j++;
                                if(j<r.size()){
                                    temp = r[j];
                                    if(!(temp>='A'&&temp<='Z')){
                                        gmp[r[i]].insert(temp);
                                        break;
                                    }
                                }
                                else{
                                    for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
                                    break;
                                }
                            }
                            else{
```

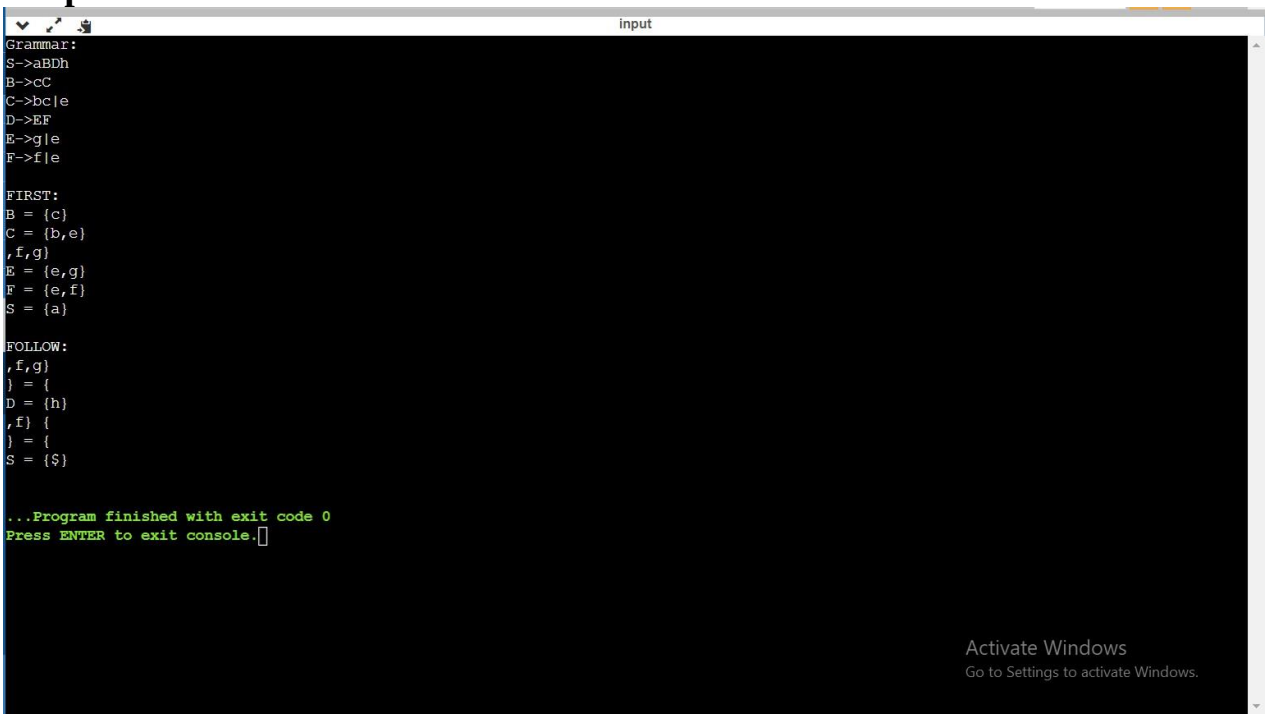
```
                        for(auto g : fmp[temp]){
                            gmp[r[i]].insert(g);
                        }
                        break;
                    }
                }
            }
        }
        if(r[r.size()-1]>='A'&&r[r.size()-1]<='Z'){
            for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
        }
    }
}

cout<<'\n';
cout<<"FOLLOW: "<<'\n';
for(auto q : gmp){
    string ans = "";
    ans += q.first;
    ans += " = {";
    for(char r : q.second){
        ans += r;
        ans += ',';
    }
    ans.pop_back();
    ans+="}";
    cout<<ans<<'\n';
}
return 0;
}
```

Input (inputfirstfollow.txt):

```
S->aBDh
B->cC
C->bc|e
D->EF
E->g|e
F->f|e
```

Output:



Conclusion : In this practical we implemented a program that simulate the process of finding first and follow of a grammar.