Experiment – 6

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

Description -

FOLLOW

Follow(X) to be the set of terminals that can appear immediately to the right of Non-Terminal X in some sentential form.

Rules to compute Follow set:

To compute FOLLOW(A) for all non-terminals A, apply the following rules until nothing can be added to any FOLLOW set.

1. Place $ in FOLLOW(S), where S is the start symbol and $ is the input right endmarker.

2. If there is a production A->αBβ, then everything in FIRST(β) except for ε is placed in FOLLOW(B).

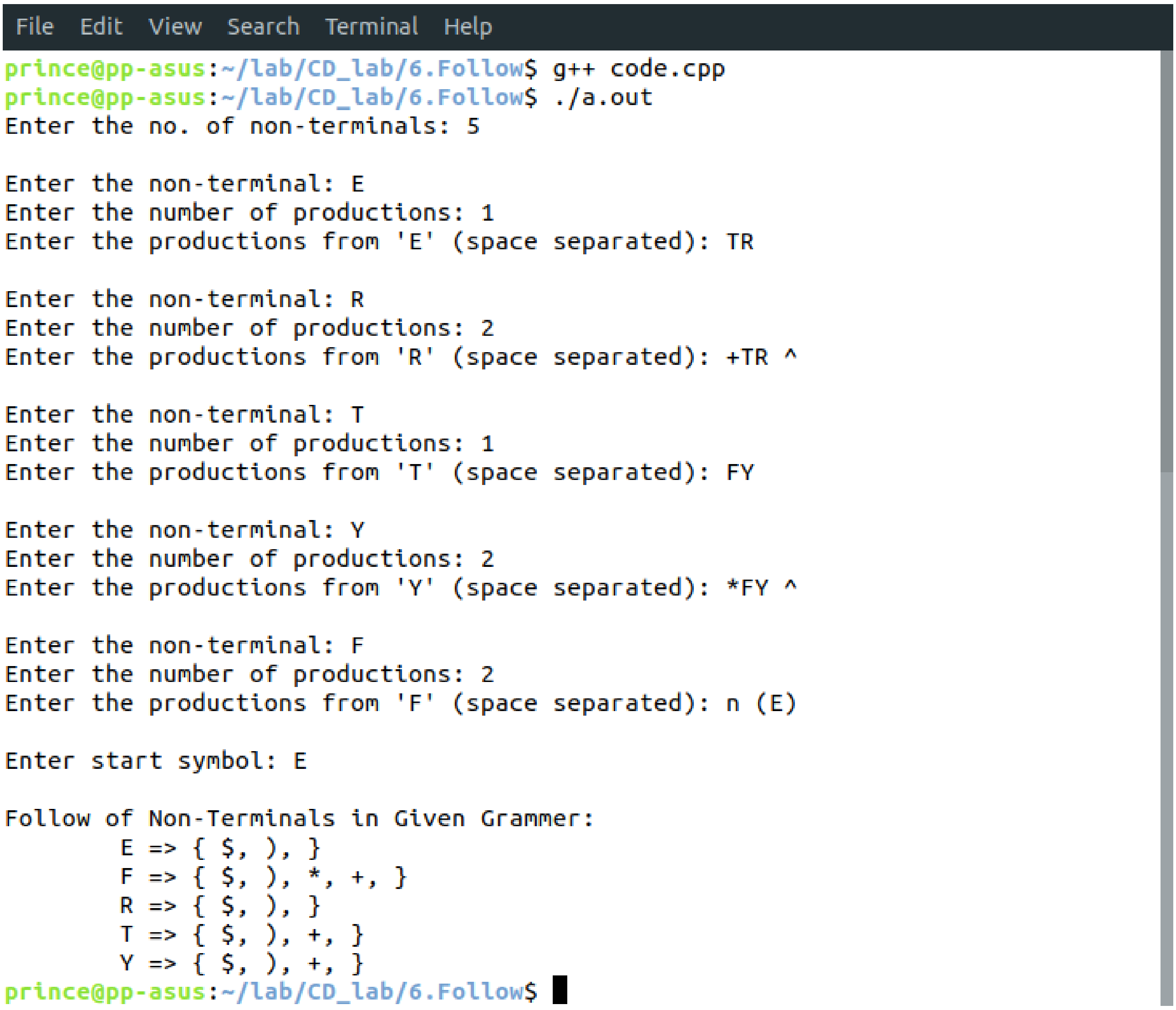
3. If there is a production A->αB, or a production A->αBβ where FIRST(β) contains ε, then everything in FOLLOW(A) is in FOLLOW(B).

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;
}

void calcFollow(string nonTerminal,
                set < string > &followTemp,
                string startingSymbol) {
    for(auto p : grammar){
        if(nonTerminal==startingSymbol){
            followTemp.insert("$");
        }

        string non_terminal = p.first;
        vector<string> prods = grammar[non_terminal];

        for(auto pd:prods){
            string prod = pd;
            int pos = 0, sz = prod.size();
            pos = prod.find(nonTerminal, 0);

            while(pos<sz && pos!=-1){
                if(pos==sz){
                    break;
                }
                if(pos==prod.size()-1 && non_terminal==nonTerminal){
                    break;
                }
                if(pos==prod.size()-1){
                    set < string > temp;
                    calcFollow(non_terminal, temp, startingSymbol);
                    setUnion(followTemp, temp);
                    break;
                }
                if(pos+1<sz && isCapital(prod[pos+1])){
                    set<string> temp;

                    calcFirst(string(1, prod[pos+1]), temp);
                    setUnion(followTemp, temp);

                    if(ifEpsilon(temp)){
                        pos++;
                        while(ifEpsilon(temp) && pos<sz-1){
                            temp.clear();
                            calcFirst(string(1, prod[pos+1]), temp);
                            setUnion(followTemp, temp);
                            pos++;
                        }
                        if(ifEpsilon(temp)){
                            set < string > tmp;
                            calcFollow(non_terminal, tmp, startingSymbol);
                            setUnion(followTemp, tmp);
                        }
                    }else{
                        setUnion(followTemp, temp);
                    }
                }else{
                    set < string > temp;
                    temp.insert(string(1, prod[pos+1]));
                    setUnion(followTemp, temp);
                }
                pos = prod.find(nonTerminal, pos+1);
            }
        }
    }
}

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<<"\nFollow of Non-Terminals in Given Grammer: \n";
    for(auto p:grammar){
        cout<<"\t";
        set<string> followTemp;

        calcFollow(p.first, followTemp,startSymbol);
        cout<<p.first<<" => { ";
        setPrint(followTemp);
        cout<<"}"<<endl;
    }
    return 0;
}
```

Output –



```
prince@pp-asus:~/lab/CD_lab/6.Follow$ g++ code.cpp
prince@pp-asus:~/lab/CD_lab/6.Follow$ ./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

Follow of Non-Terminals in Given Grammer:
        E => { $, ), }
        F => { $, ), *, +, }
        R => { $, ), }
        T => { $, ), +, }
        Y => { $, ), +, }
prince@pp-asus:~/lab/CD_lab/6.Follow$ 
```

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.