Practical - 3

2CS701 – Compiler Construction

Harshit Gajipara

19bce059

**Aim:**

Write a program to find first() and follow() set for each non-terminal of given grammar.

**Code:**

/\*

Write a program to find first(), and follow() set for each non-terminal of given grammar.

\*/

#include <bits/stdc++.h>

using namespace std;

int no\_of\_terminals, no\_of\_non\_terminals, no\_of\_productions;

string \*terminals, \*non\_terminals, starting\_symbol, \*productions;

map<string, vector<string>> productions\_map;

map<string, set<string>> FIRST;

map<string, set<string>> FOLLOW;

template <typename T>

set<T> getUnion(const set<T> &a, const set<T> &b)

{

    set<T> result = a;

    result.insert(b.begin(), b.end());

    return result;

}

string getString(char x)

{

    string s(1, x);

    return s;

}

vector<string> split\_production(string input, string delimiter)

{

    size\_t pos = 0;

    string token;

    vector<string> prods;

    while((pos = input.find(delimiter)) != string::npos)

    {

        token = input.substr(0, pos);

        prods.push\_back(token);

        input.erase(0, pos + delimiter.length());

    }

    prods.push\_back(input);

    return prods;

}

bool is\_in\_array(string s, string \*array, int size)

{

    for (int i = 0; i < size; i++)

    {

        if (array[i] == s)

            return true;

    }

    return false;

}

set<string> first(string s)

{

    set<string> first\_;

    if (is\_in\_array(s, non\_terminals, no\_of\_non\_terminals))

    {

        vector<string> alternatives = productions\_map[s];

        for (int i = 0; i < alternatives.size(); ++i)

        {

            string temp = alternatives[i];

            set<string> first\_2 = first(temp);

            first\_ = getUnion(first\_, first\_2);

        }

    }

    else if (is\_in\_array(s, terminals, no\_of\_terminals))

    {

        first\_ = {s};

    }

    else if (s == "" || s == "@")

    {

        first\_ = {"@"};

    }

    else

    {

        set<string> first\_2 = first(getString(s[0]));

        if (first\_2.find("@") != first\_2.end())

        {

            int i = 1;

            while (first\_2.find("@") != first\_2.end())

            {

                set<string> ne = first\_2;

                ne.erase("@");

                first\_ = getUnion(first\_, ne);

                if (is\_in\_array(s.substr(i), terminals, no\_of\_terminals))

                {

                    set<string> t = {s.substr(i)};

                    first\_ = getUnion(first\_, t);

                    break;

                }

                else if (s.substr(i) == "")

                {

                    set<string> t = {"@"};

                    first\_ = getUnion(first\_, t);

                    break;

                }

                ne = first(s.substr(i));

                ne.erase("@");

                first\_ = getUnion(first\_, ne);

                i++;

            }

        }

        else

        {

            first\_ = getUnion(first\_, first\_2);

        }

    }

    return first\_;

}

set<string> follow(string nT)

{

    set<string> follow\_;

    if (nT == starting\_symbol)

    {

        set<string> dollar = {"$"};

        follow\_ = getUnion(follow\_, dollar);

    }

    map<string, vector<string>>::iterator itr;

    for (itr = productions\_map.begin(); itr != productions\_map.end(); ++itr)

    {

        string nt = itr->first;

        vector<string> rhs = itr->second;

        for (auto alt = rhs.begin(); alt != rhs.end(); ++alt)

        {

            for (int i = 0; i < (\*alt).length(); i++)

            {

                if (nT == getString((\*alt)[i]))

                {

                    string following\_str = (\*alt).substr(i + 1);

                    if (following\_str == "")

                    {

                        if (nT == nt)

                        {

                            continue;

                        }

                        else

                        {

                            follow\_ = getUnion(follow\_, follow(nt));

                        }

                    }

                    else

                    {

                        set<string> follow\_2 = first(following\_str);

                        if (follow\_2.find("@") != follow\_2.end())

                        {

                            set<string> t = follow\_2;

                            t.erase("@");

                            follow\_ = getUnion(follow\_, t);

                            follow\_ = getUnion(follow\_, follow(nt));

                        }

                        else

                        {

                            follow\_ = getUnion(follow\_, follow\_2);

                        }

                    }

                }

            }

        }

    }

    return follow\_;

}

void scaninput()

{

    cout << "Enter no. of terminals : ";

    cin >> no\_of\_terminals;

    terminals = new string[no\_of\_terminals];

    cout << "Enter the terminals :" << endl;

    for (int i = 0; i < no\_of\_terminals; i++)

        cin >> terminals[i];

    cout << "\nEnter no. of non terminals : ";

    cin >> no\_of\_non\_terminals;

    non\_terminals = new string[no\_of\_non\_terminals];

    cout << "Enter the non terminals :" << endl;

    for (int i = 0; i < no\_of\_non\_terminals; i++)

        cin >> non\_terminals[i];

    cout << "\nEnter the starting symbol : ";

    cin >> starting\_symbol;

    cout << "\nEnter the number of productions : ";

    cin >> no\_of\_productions;

    productions = new string[no\_of\_productions];

    cout << "Enter the productions : \n";

    for (int i = 0; i < no\_of\_productions; i++)

    {

        cin >> productions[i];

        vector<string> temp = split\_production(productions[i], "->");

        vector<string> temp2 = split\_production(temp[1], "|");

        productions\_map.insert(pair<string, vector<string>>(temp[0], temp2));

    }

    cout << "\nProductions : \n";

    for (auto itr = productions\_map.begin(); itr != productions\_map.end(); ++itr)

    {

        cout << itr->first << " -> ";

        for (auto i = itr->second.begin(); i != itr->second.end(); ++i)

            cout << \*i << " ";

        cout << endl;

    }

}

void calculate\_first\_and\_follow()

{

    for (int i = 0; i < no\_of\_non\_terminals; i++)

        FIRST[non\_terminals[i]] = getUnion(FIRST[non\_terminals[i]], first(non\_terminals[i]));

    set<string> dollar = {"$"};

    FOLLOW[starting\_symbol] = getUnion(FOLLOW[starting\_symbol], dollar);

    for (int i = 0; i < no\_of\_non\_terminals; i++)

        FOLLOW[non\_terminals[i]] = getUnion(FOLLOW[non\_terminals[i]], follow(non\_terminals[i]));

}

void print\_first\_and\_follow()

{

    cout << "\nNon Terminals \t First \t\tFollow" << endl;

    for (int i = 0; i < no\_of\_non\_terminals; i++)

    {

        cout << non\_terminals[i] << " \t\t ";

        for (auto itr = FIRST[non\_terminals[i]].begin(); itr != FIRST[non\_terminals[i]].end(); ++itr)

            cout << \*itr << " ";

        cout << "\t\t";

        for (auto itr = FOLLOW[non\_terminals[i]].begin(); itr != FOLLOW[non\_terminals[i]].end(); ++itr)

            cout << \*itr << " ";

        cout << endl;

    }

}

int main()

{

    scaninput();

    // initialize an empty set of strings of first and follow for each non terminal

    for (int i = 0; i < no\_of\_non\_terminals; i++)

    {

        FIRST[non\_terminals[i]] = {};

        FOLLOW[non\_terminals[i]] = {};

    }

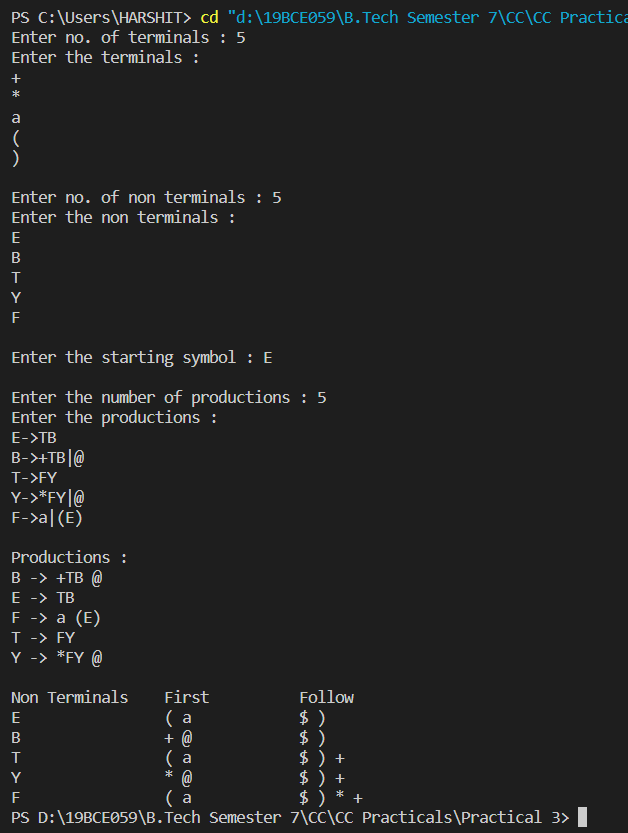
    calculate\_first\_and\_follow();

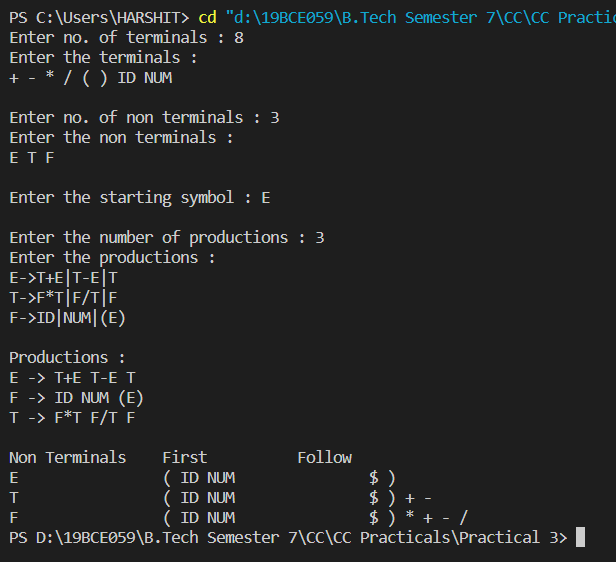
    print\_first\_and\_follow();

    return 0;

}

**Output:**





**Conclusion:**

In this practical, we learnt how to implement cpp program to find first and follow of given grammar using map and set to store production rules. We can also use first and find with grammar containing null production also.