Practical - 3

2CS701 – Compiler Construction

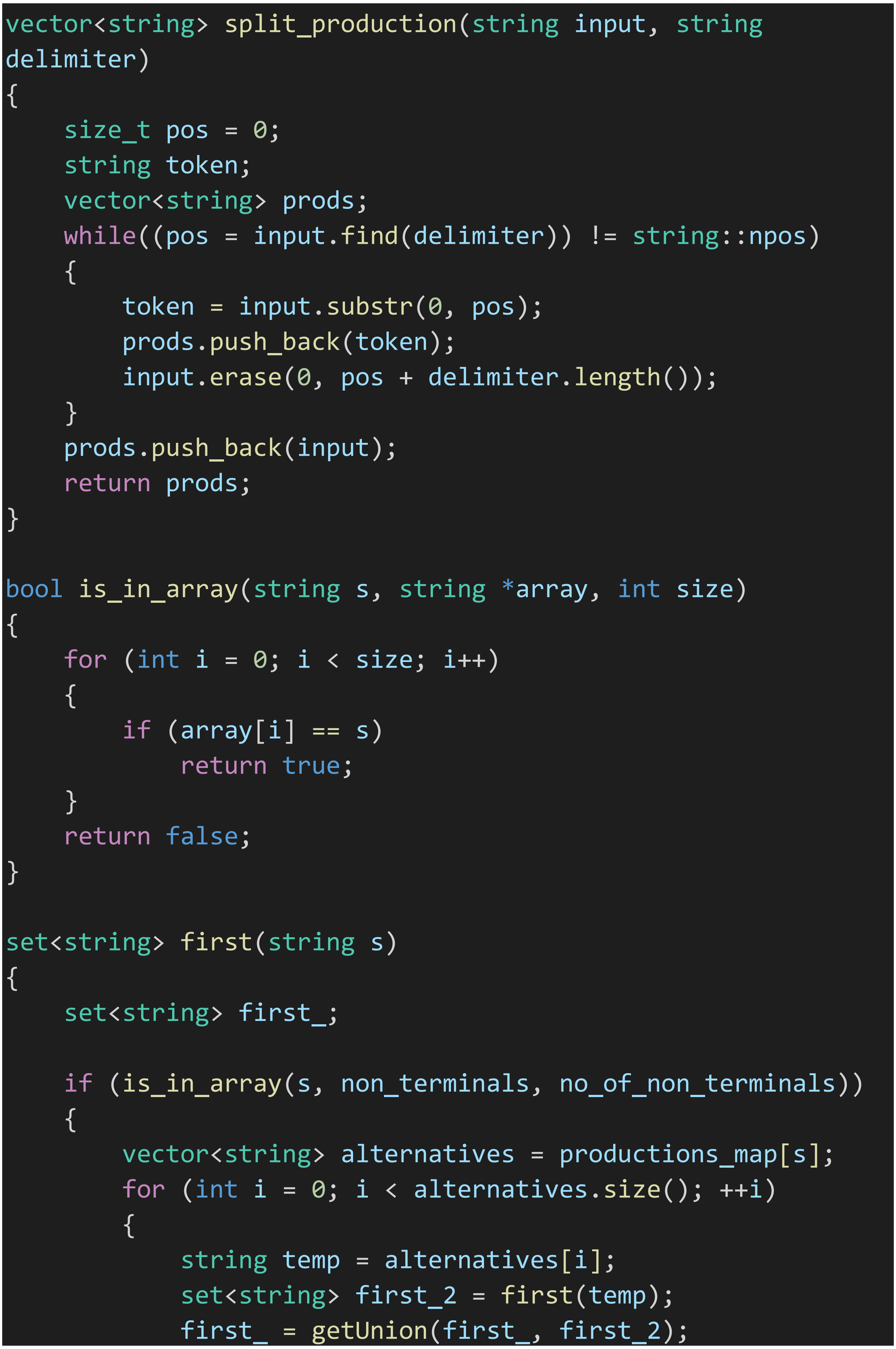
Harshit Gajipara
19BCE059

Aim:

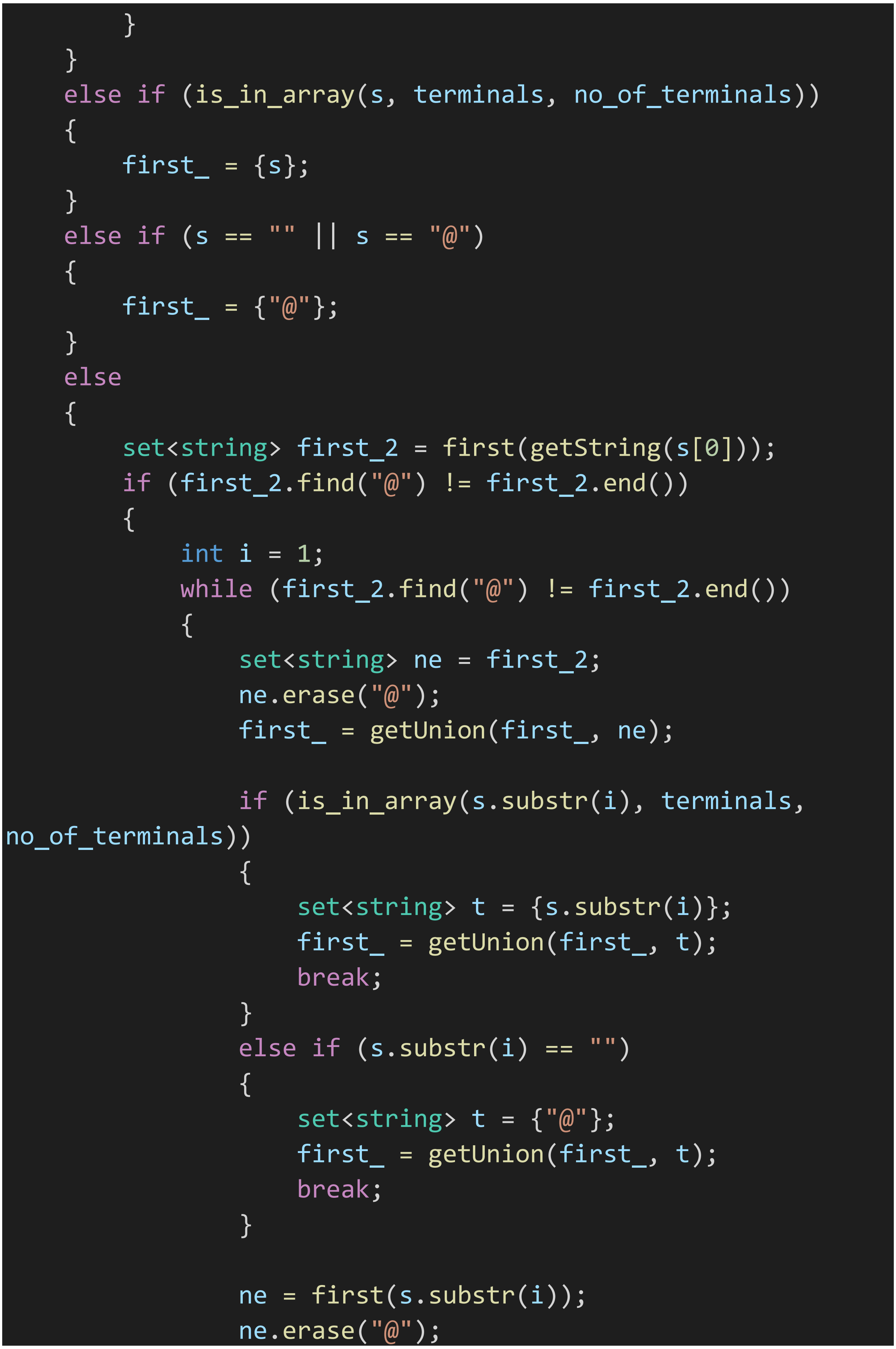
Write a program to find first() and follow() set for each non-terminal of given grammar.

Code:

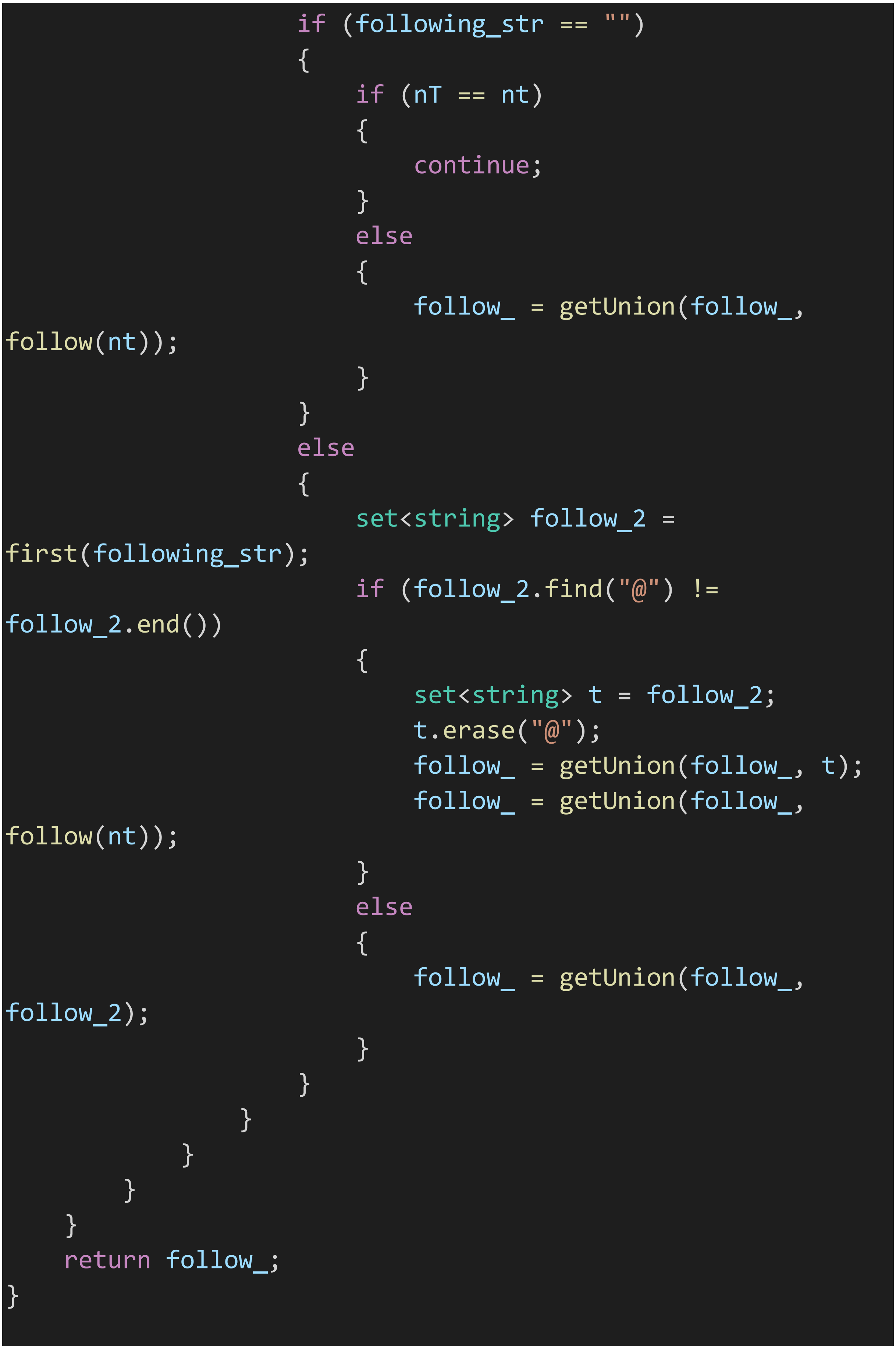
```
/*
Write a program to find first(), and follow() set for each
non-terminal of given grammar.
*/

#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;
}
```



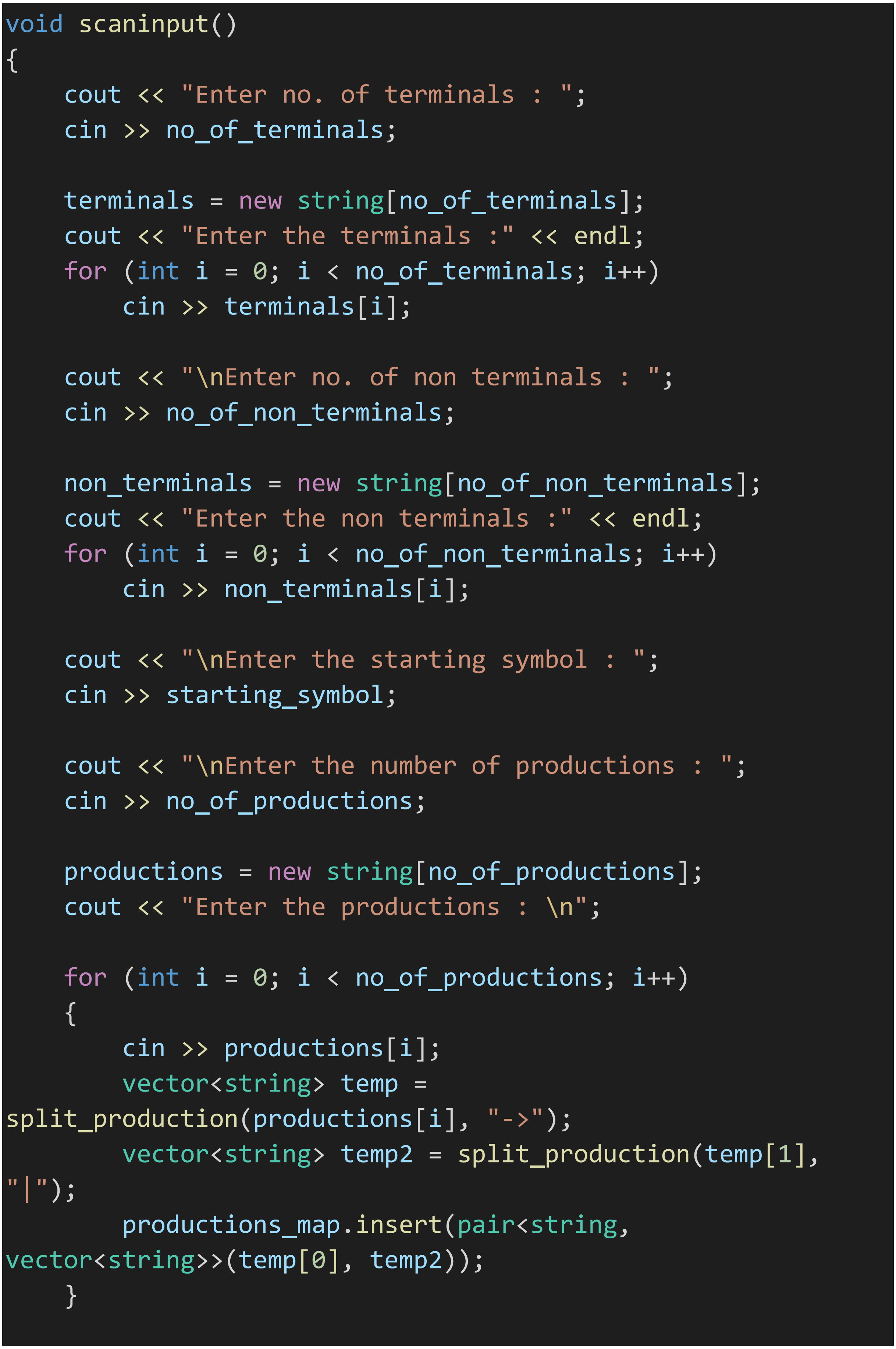
```
vector<string> split_production(string input, string
delimiter)
{
    size_t pos = 0;
    string token;
    vector<string> prods;
    while((pos = input.find(delimiter)) != string::npos)
    {
        token = input.substr(0, pos);
        prods.push_back(token);
        input.erase(0, pos + delimiter.length());
    }
    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);
```

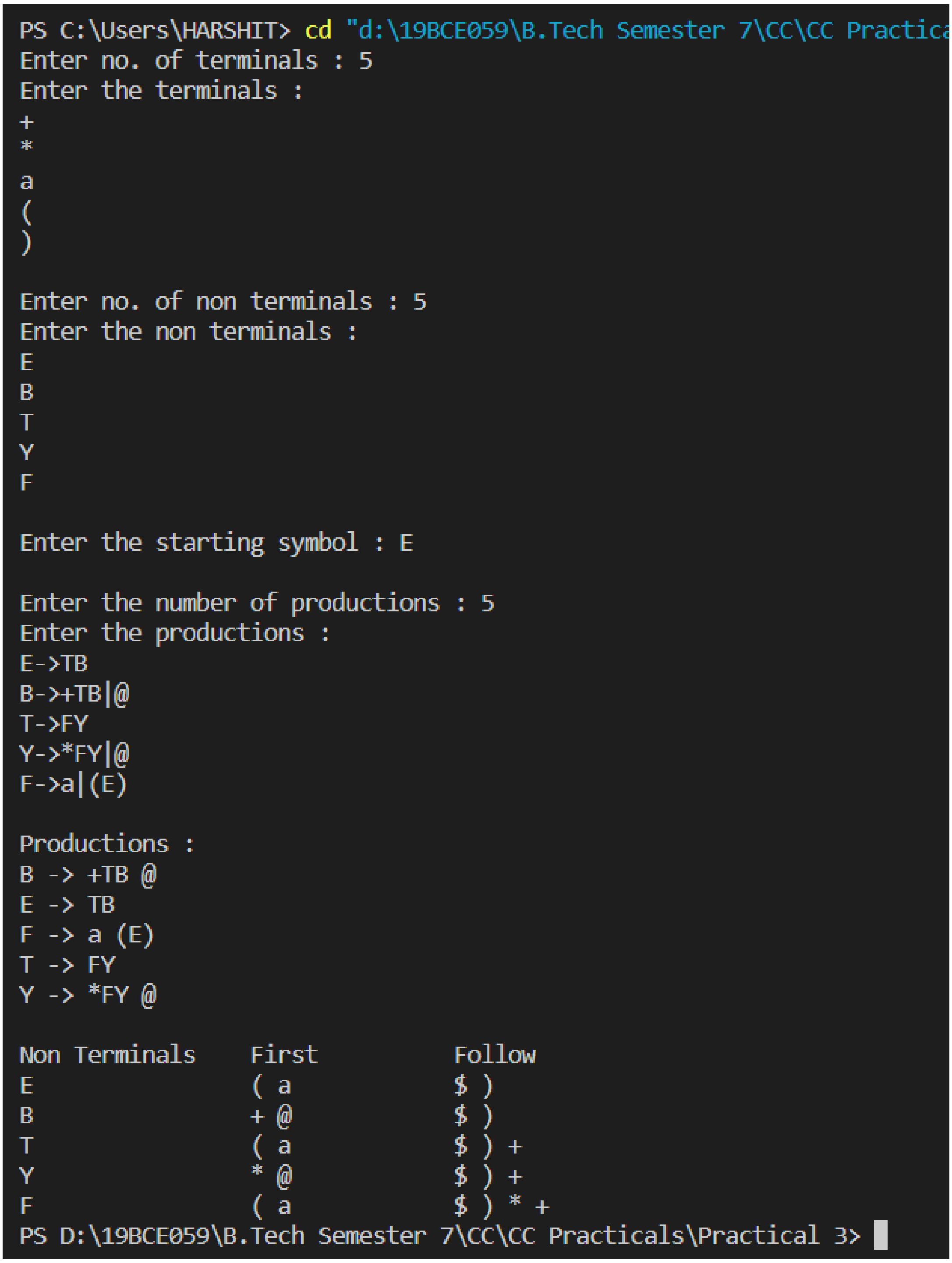


```
                    if (following_str == "")
                    {
                        if (nT == nt)
                        {
                            continue;
                        }
                        else
                        {
                            follow_ = getUnion(follow_,
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_;
}
```

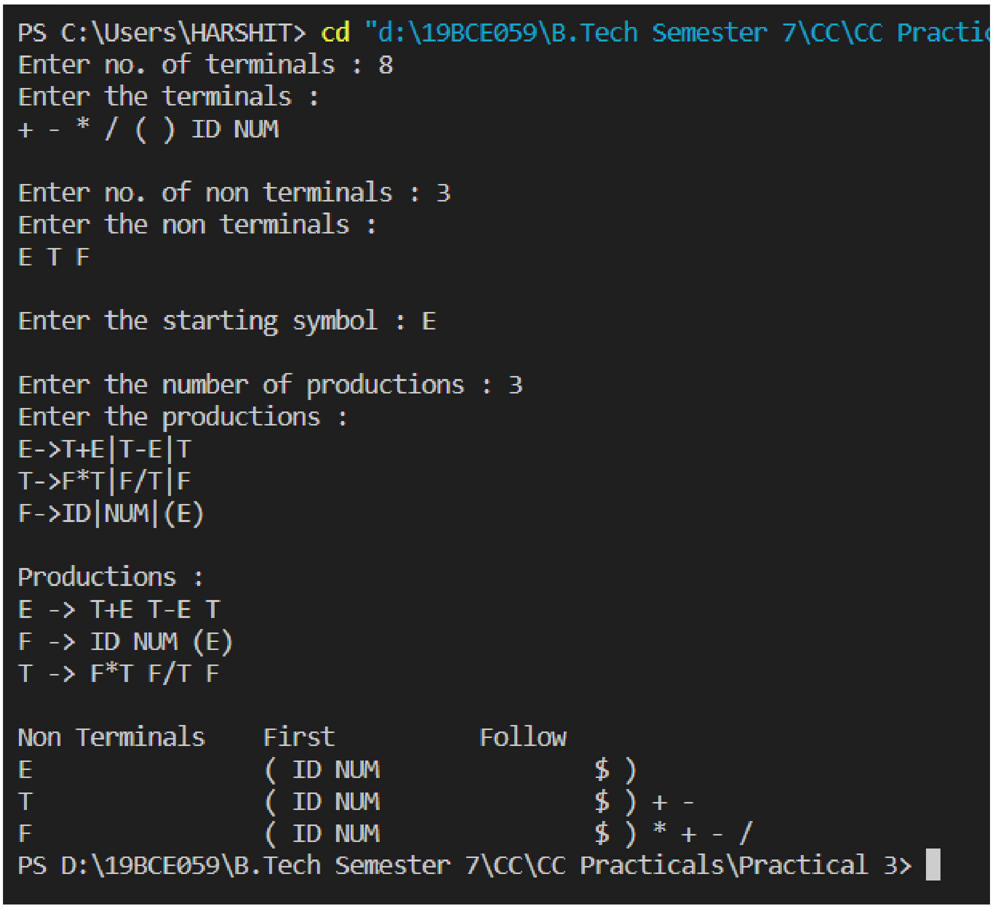


```
void scaninput()
{
    cout << "Enter no. of terminals : ";
    cin >> no_of_terminals;

    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:



```
PS C:\Users\HARSHIT> cd "d:\19BCE059\B.Tech Semester 7\CC\CC Practica
Enter no. of terminals : 5
Enter the terminals :
+
*
a
(
)

Enter no. of non terminals : 5
Enter the non terminals :
E
B
T
Y
F

Enter the starting symbol : E

Enter the number of productions : 5
Enter the productions :
E->TB
B->+TB|@
T->FY
Y->*FY|@
F->a|(E)

Productions :
B -> +TB @
E -> TB
F -> a (E)
T -> FY
Y -> *FY @

Non Terminals    First          Follow
E                ( a            $ )
B                + @            $ )
T                ( a            $ ) +
Y                * @            $ ) +
F                ( a            $ ) * +
PS D:\19BCE059\B.Tech Semester 7\CC\CC Practicals\Practical 3>
```



```
PS C:\Users\HARSHIT> cd "d:\19BCE059\B.Tech Semester 7\CC\CC Practi
Enter no. of terminals : 8
Enter the terminals :
+ - * / ( ) ID NUM

Enter no. of non terminals : 3
Enter the non terminals :
E T F

Enter the starting symbol : E

Enter the number of productions : 3
Enter the productions :
E->T+E|T-E|T
T->F*T|F/T|F
F->ID|NUM|(E)

Productions :
E -> T+E T-E T
F -> ID NUM (E)
T -> F*T F/T F

Non Terminals    First          Follow
E                ( ID NUM               $ )
T                ( ID NUM               $ ) + -
F                ( ID NUM               $ ) * + - /
PS D:\19BCE059\B.Tech Semester 7\CC\CC Practicals\Practical 3>
```

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.