**Practical - 7**

**Code:**#include <bits/stdc++.h>

using namespace std;

vector<pair<char, string>> grammar;

map<char, set<string>> first;

map<char, set<string>> follow;

void computeFirst(char nonTerminal)

{

for (auto &rule : grammar)

{

if (rule.first == nonTerminal)

{

if (isupper(rule.second[0]))

{

computeFirst(rule.second[0]);

first[nonTerminal].insert(first[rule.second[0]].begin(), first[rule.second[0]].end());

if (first[rule.second[0]].count("#") > 0 && rule.second.size() > 1)

{

char nextSymbol = rule.second[1];

if (isupper(nextSymbol))

{

computeFirst(nextSymbol);

first[nonTerminal].insert(first[nextSymbol].begin(), first[nextSymbol].end());

}

else

{

first[nonTerminal].insert(string(1, nextSymbol));

}

}

}

else if (rule.second[0] == '#')

{

first[nonTerminal].insert("#");

}

else

{

first[nonTerminal].insert(string(1, rule.second[0]));

}

}

}

}

void computeFollow(char nonTerminal)

{

if (nonTerminal == 'S')

{

follow[nonTerminal].insert("$");

}

for (auto &rule : grammar)

{

size\_t pos = rule.second.find(nonTerminal);

while (pos != string::npos)

{

if (pos + 1 < rule.second.size())

{

char nextSymbol = rule.second[pos + 1];

if (isupper(nextSymbol))

{

for (auto i : first[nextSymbol])

if (i != "#")

follow[nonTerminal].insert(i);

}

else

{

follow[nonTerminal].insert(string(1, nextSymbol));

}

}

else

{

if (rule.first != nonTerminal)

{

computeFollow(rule.first);

follow[nonTerminal].insert(follow[rule.first].begin(), follow[rule.first].end());

}

}

pos = rule.second.find(nonTerminal, pos + 1);

}

}

}

int main()

{

grammar.push\_back({'S', "ABC"});

grammar.push\_back({'S', "D"});

grammar.push\_back({'A', "a"});

grammar.push\_back({'A', "#"});

grammar.push\_back({'B', "b"});

grammar.push\_back({'B', "#"});

grammar.push\_back({'C', "(S)"});

grammar.push\_back({'C', "c"});

grammar.push\_back({'D', "AC"});

for (auto &rule : grammar)

{

computeFirst(rule.first);

}

for (auto &rule : grammar)

{

computeFollow(rule.first);

}

cout << "FIRST sets:" << endl;

for (auto &entry : first)

{

cout << "FIRST(" << entry.first << ") = { ";

for (auto &s : entry.second)

cout << s << ",";

cout << "}" << endl;

}

cout << "FOLLOW sets:" << endl;

for (auto &entry : follow)

{

cout << "FOLLOW(" << entry.first << ") = { ";

for (auto &s : entry.second)

cout << s << ", ";

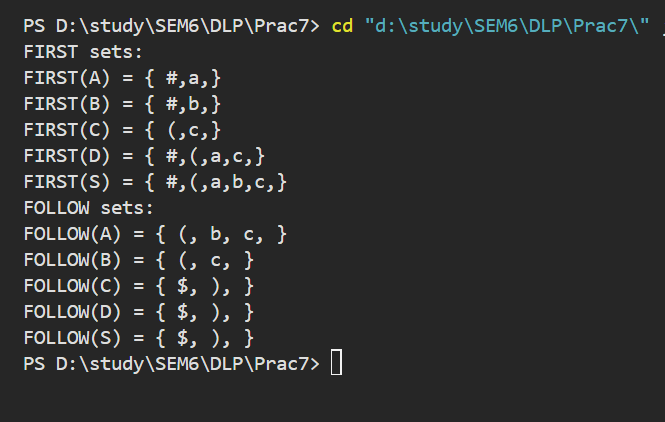
cout << "}" << endl;

}

return 0;

}

**Output Screenshot:**

****