```
#include <iostream>
#include <map>
#include <string>
#include <vector>
#include <sstream>


using namespace std;

int main()
{
        int total_productions=0;
        cout << "Enter the total number of productions : ";
        cin >> total_productions;
        cout << endl;
        string cfg[total_productions];

        map<char, string> first_set;
        map<char, string> follow_set;
        map<string, string> parsing_table;


        cout << "Enter the productions of the grammer (ex: S-bAc) : \n";
        for (int i = 0; i < total_productions; ++i)
        {
                cout << "Production [" << (i+1) << "] : ";
                cin >> cfg[i];
                first_set[cfg[i][0]] = "";
                follow_set[cfg[i][0]] = "";
        }


        //first
        int start_index = 2;
        for(int i = total_productions-1; i >= 0; i--)
        {
                string temp_first_symbols;
                stringstream ss;
                string temp_symbol_of_production;

                if(!((int)cfg[i][start_index] >= 65 && (int)cfg[i][start_index] <= 90))
                {
                        temp_first_symbols = first_set[cfg[i][0]];
                        ss << cfg[i][start_index];
                        ss >> temp_symbol_of_production;
                        temp_first_symbols.append(temp_symbol_of_production);
                        first_set[cfg[i][0]] = temp_first_symbols;
                }
                else
                {
                        temp_first_symbols = first_set[cfg[i][0]];
                        temp_first_symbols.append(first_set[cfg[i][start_index]]);
                        first_set[cfg[i][0]] = temp_first_symbols;

                        unsigned found = first_set[cfg[i][start_index]].find('@');
                        while(found != string::npos)
                        {
                                start_index++;
                                temp_first_symbols = first_set[cfg[i][0]];
                                temp_first_symbols.append(first_set[cfg[i][start_index]]);
                                first_set[cfg[i][0]] = temp_first_symbols;
                        }
                        start_index = 2;
                }
        }

        string all_n_term = "";
        for (int i = 0; i < total_productions; ++i)
        {
                unsigned found = all_n_term.find(cfg[i][0]);
                if(found == string::npos)
                {
                        stringstream ss;
                        string temp;
                        ss << cfg[i][0];
                        ss >> temp;
                        all_n_term.append(temp);
                }
        }


        //follow
        follow_set[cfg[0][0]].append("$");      //rule for start symbol

        for (int i = 0; i < all_n_term.size(); ++i)
        {
                char temp_n_term = all_n_term[i];
                vector<string> temp_prod_set;
                for(int j = 0; j < total_productions; j++)
                {
                        if(temp_n_term != cfg[j][0])
                        {
                                unsigned found = cfg[j].find(temp_n_term,2);
                                if(found != string::npos)
                                {
                                        temp_prod_set.push_back(cfg[j]);
                                }
                        }
                }
                while(!temp_prod_set.empty())
                {
                        string temp_prod = temp_prod_set.back();
                        temp_prod_set.pop_back();
                        if(temp_prod[0] == temp_prod[temp_prod.size()-1])
                                continue;
                        else
                        {
                                unsigned found = temp_prod.find(temp_n_term);
                                if(found != (temp_prod.size()-1))       //that is it is of the form A-
> bB(beta) where (beta) is a single term
                                {
                                        string beta = temp_prod.substr(found+1, (temp_prod.size() -
(found +1)));
                                        for(int k = 0; k < beta.size(); k++)
                                        {
                                                stringstream ss;
                                                string temp;
                                                string temp_first_symbols;
                                                bool break_loop = false;

                                                if(!((int)beta[k] >= 65 && (int)beta[k] <= 90))
                                                {
                                                        ss << beta
[k];
                                                        ss >> temp;
                                                        follow_set[temp_n_term].append(temp);
                                                        break_loop = true;
                                                }
                                                else
                                                {
                                                        bool has_emty_symbol = false;   //has @
                                                        unsigned found = first_set[beta[k]].find('@');
                                                        if(found != string::npos)
                                                        {
                                                                temp_first_symbols = first_set[beta
[k]];
                                                                temp_first_symbols.erase(found, 1);
                                                                follow_set[temp_n_term].append
(temp_first_symbols);
                                                                follow_set[temp_n_term].append
(follow_set[temp_prod[0]]);
                                                        }
                                                        else
                                                        {
                                                                follow_set[temp_n_term].append
(first_set[beta[k]]);
                                                                break_loop = true;
                                                        }
                                                }
                                                if(break_loop)
                                                        break;
                                        }
                                }
                                else
                                {
                                        follow_set[temp_n_term].append(follow_set[temp_prod[0]]);
                                }
                        }
                }
        }

        //parsing table
        for(int i = 0; i < all_n_term.size(); i++)
        {
                char temp_n_term = all_n_term[i];
                vector<string> temp_prod_set;
                for(int j = 0; j < total_productions; j++)
                {
                        if(temp_n_term == cfg[j][0])
                        {
                                temp_prod_set.push_back(cfg[j]);
                        }
                }
                while(!temp_prod_set.empty())
                {
                        string temp_prod = temp_prod_set.back();
                        temp_prod_set.pop_back();
                        string all_temp_term = "";
                        for(int k = 2; k < temp_prod.size(); k++)
                        {
                                if(temp_prod[k] == '@')
                                {
                                        all_temp_term.append(follow_set[temp_prod[0]]);
                                        break;
                                }
                                else if(!((int)temp_prod[k] >= 65 && (int)temp_prod[k] <= 90))
                                {
                                        string temp;
                                        stringstream ss;
                                        ss << temp_prod[k];
                                        ss >> temp;
                                        all_temp_term.append(temp);
                                        break;
                                }
                                else
                                {
                                        unsigned found = first_set[temp_prod[k]].find('@');
                                        if(found != string::npos)
                                        {
                                                string temp_first_symbols;
                                                temp_first_symbols = first_set[temp_prod[k]];
                                                temp_first_symbols.erase(found, 1);
                                                all_temp_term.append(temp_first_symbols);
                                        }
                                        else
                                        {
                                                all_temp_term.append(first_set[temp_prod[k]]);
                                                break;
                                        }
                                }
                        }
                        for(int l = 0; l < all_temp_term.size(); l++)
                        {
                                stringstream ss;
                                ss << temp_prod[0] << all_temp_term[l];
                                string temp = ss.str();
                                parsing_table[temp] = temp_prod;
                        }
                }
        }

        //string parsing
        string input_string;
        cout << "Enter the input string : ";
        cin >> input_string;
        input_string.append("$");
        int ip = 0;

        vector<char> stack;
        stack.push_back('$');   //right end marker at the bottom of the stack
        stack.push_back(cfg[0][0]);     //start symbol of the grammer at the top in the begining
        char stack_top = stack.back();

        while(stack_top != '$')
        {
                stringstream ss;
                ss << stack_top << input_string[ip];
                string temp_table_index = ss.str();
                map<string, string>::iterator it;
                bool present = false;
                it = parsing_table.find(temp_table_index);

                if(it != parsing_table.end())
                        present = true;

                if(temp_table_index[0] == temp_table_index[1])
                {
                        stack.pop_back();
                        ip = ip + 1;
                        stack_top = stack.back();
                }
                else if(present)
                {
                        string temp_prod = it->second;
                        cout << "OUTPUT : " << temp_prod << endl;
                        stack.pop_back();
                        if(temp_prod[2] != '@')
                        {
                                for(int i = temp_prod.size()-1; i >= 2; i--)
                                {
                                        stack.push_back(temp_prod[i]);
                                }
                        }
                        stack_top = stack.back();
                }
                else
                        break;  //all types of error condition covered here
        }

        if(stack_top == '$' && input_string[ip] == '$')
                cout << "Input string accepted.\n" << endl;
        else
                cout << "Input string NOT accepted.\n" << endl;
        return 0;
}
```