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

using namespace std;

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

        cout << endl;
        string operator_grammar[total_productions];


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

        string all_operators = "";
        for (int i = 0; i < total_productions; ++i)
        {
                for(int j = 2; j < operator_grammar[i].size(); j++)
                {
                        if(!((int)operator_grammar[i][j] >= 65 && (int)operator_grammar[i][j] <= 90))
                        {
                                stringstream ss;
                                string temp;

                                ss << operator_grammar[i][j];
                                temp = ss.str();

                                all_operators.append(temp);
                        }
                }
        }

        all_operators.append("$");      //appending the right end marker at the right end

        map<string, char> op_pred_matrx;
        map<string, char>::iterator it;

        cout << "\n:::::: Enter the Operator Precendence Matrix ::::::\n";
        for (int i = 0; i < all_operators.size(); ++i)
        {
                for(int j = 0; j < all_operators.size(); ++j)
                {
                        stringstream ss;
                        string matrx_index;
                        char temp_op;

                        ss << all_operators[i] << all_operators[j];
                        matrx_index = ss.str();

                        cout << "Matrix Entry for [" << matrx_index[0] << "][" << matrx_index[1] << "]
is (enter # for error entry) : ";
                        cin >> temp_op;
                        op_pred_matrx[matrx_index] = temp_op;
                }
        }


        cout << "\n:::::: Operator Precendence Matrix ::::::\n\n";
        for(it = op_pred_matrx.begin(); it != op_pred_matrx.end(); it++)
                cout << it->first << "  " << it->second << endl;
        string input_string;
        cout << "\nEnter 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
        char stack_top = stack.back();


        cout << ":::::: ACTIONS ::::::" << endl;
        while(true)
        {
                if(stack_top == '$' && input_string[ip] == '$')
                {
                        cout << "ACCEPTED" << endl;
                        break;
                }
                else
                {
                        stringstream ss;
                        string matrx_index;
                        char temp_op;

                        ss << stack.back() << input_string[ip];
                        matrx_index = ss.str();

                        temp_op = op_pred_matrx[matrx_index];

                        if(temp_op == '<' || temp_op == '=')    //SHIFT
                        {
                                cout << "SHIFT [" << input_string[ip] << "]" << endl;
                                stack.push_back(input_string[ip]);
                                stack_top = stack.back();
                                ip = ip + 1;
                        }
                        else if(temp_op == '>') //REDUCE
                        {
                                char new_temp_op;
                                do
                                {
                                        for (int i = 0; i < total_productions; ++i)
                                        {
                                                unsigned found  = operator_grammar[i].find(stack_top);
                                                if(found != string::npos)
                                                {
                                                        cout << "REDUCTION : " << operator_grammar[i]
<< endl;

                                                        break;
                                                }
                                        }
                                        if(stack_top != '$')
                                                stack.pop_back();

                                        stack_top = stack.back();

                                        stringstream ss_new;
                                        string matrx_index_new;

                                        ss_new << stack_top << input_string[ip];
                                        matrx_index_new = ss_new.str();

                                        new_temp_op = op_pred_matrx[matrx_index_new];

                                }while(new_temp_op != '<' && new_temp_op != '#');
                        }
                        else
                        {
                                cout << "ERROR!!!" << endl;
                                break;
                        }
                }
        }

        return 0;
}
```