**Week 9 Assignment**

**Problem**

Implementation of Shift Reduce parser using C for the following grammar and illustrate the

parser’s actions for a valid and an invalid string.

E→E+E

E→E\*E

E→(E)

E→d

**Program**

#include <iostream>

#include <vector>

#include <map>

#include <string>

#include <stack>

#include <set>

#include <algorithm>

using namespace std;

class ShiftReduceParser

{

private:

    map<pair<char, char>, char> parsingTable;

    vector<pair<char, string>> productions;

    set<char> nonTerminals;

    int noOfProductions;

public:

    ShiftReduceParser()

    {

        parsingTable[{'+', '+'}] = '>';

        parsingTable[{'+', '\*'}] = '<';

        parsingTable[{'+', '('}] = '<';

        parsingTable[{'+', ')'}] = '>';

        parsingTable[{'+', 'd'}] = '<';

        parsingTable[{'+', '$'}] = '>';

        parsingTable[{'\*', '+'}] = '>';

        parsingTable[{'\*', '\*'}] = '>';

        parsingTable[{'\*', '('}] = '<';

        parsingTable[{'\*', ')'}] = '>';

        parsingTable[{'\*', 'd'}] = '<';

        parsingTable[{'\*', '$'}] = '>';

        parsingTable[{'(', '+'}] = '<';

        parsingTable[{'(', '\*'}] = '<';

        parsingTable[{'(', '('}] = '<';

        parsingTable[{'(', ')'}] = '=';

        parsingTable[{'(', 'd'}] = '<';

        parsingTable[{'(', '$'}] = 'x';

        parsingTable[{')', '+'}] = '>';

        parsingTable[{')', '\*'}] = '>';

        parsingTable[{')', '('}] = 'x';

        parsingTable[{')', ')'}] = '>';

        parsingTable[{')', 'd'}] = 'x';

        parsingTable[{')', '$'}] = '>';

        parsingTable[{'d', '+'}] = '>';

        parsingTable[{'d', '\*'}] = '>';

        parsingTable[{'d', '('}] = 'x';

        parsingTable[{'d', ')'}] = '>';

        parsingTable[{'d', 'd'}] = 'x';

        parsingTable[{'d', '$'}] = '>';

        parsingTable[{'$', '+'}] = '<';

        parsingTable[{'$', '\*'}] = '<';

        parsingTable[{'$', '('}] = '<';

        parsingTable[{'$', ')'}] = 'x';

        parsingTable[{'$', 'd'}] = '<';

        parsingTable[{'$', '$'}] = 'x';

    }

    void setProductions()

    {

        cout << "Enter the no of productions: ";

        cin >> noOfProductions;

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

        string production;

        char lhs;

        string rhs;

        for (int i = 0; i < noOfProductions; i++)

        {

            cin >> production;

            lhs = production[0];

            rhs = production.substr(3, production.length() - 3);

            productions.push\_back({lhs, rhs});

            nonTerminals.insert(lhs);

        }

    }

    char getTopMostTerminal(stack<char> st)

    {

        while (!st.empty())

        {

            if (find(nonTerminals.begin(), nonTerminals.end(), st.top()) == nonTerminals.end())

            {

                return st.top();

            }

            st.pop();

        }

    }

    bool parse(string input)

    {

        stack<char> st;

        st.push('$');

        input += '$';

        char stackTop;

        char inputFront;

        char topMostTerminalSymbolOnStack;

        bool oneTerminalFound = false;

        char prevTerminal;

        string toBeReduced = "";

        bool productionFound = false;

        int ptr = 0;

        while (true)

        {

            if (st.size() == 2 && st.top() == 'E' && input[ptr] == '$')

            {

                display\_stack(st);

                cout << endl;

                return true;

            }

            else

            {

                topMostTerminalSymbolOnStack = getTopMostTerminal(st);

                inputFront = input[ptr];

                if (parsingTable[{topMostTerminalSymbolOnStack, inputFront}] == '<' || parsingTable[{topMostTerminalSymbolOnStack, inputFront}] == '=')

                {

                    display\_stack(st);

                    cout << "\t < \t";

                    display\_string(input, ptr);

                    st.push(inputFront);

                    ptr++;

                }

                else if (parsingTable[{topMostTerminalSymbolOnStack, inputFront}] == '>')

                {

                    display\_stack(st);

                    cout << "\t > \t";

                    display\_string(input, ptr);

                    toBeReduced = "";

                    topMostTerminalSymbolOnStack = getTopMostTerminal(st);

                    while (st.top() != topMostTerminalSymbolOnStack)

                    {

                        toBeReduced += st.top();

                        st.pop();

                    }

                    toBeReduced += st.top();

                    prevTerminal = st.top();

                    st.pop();

                    while (true)

                    {

                        topMostTerminalSymbolOnStack = getTopMostTerminal(st);

                        while (st.top() != topMostTerminalSymbolOnStack)

                        {

                            toBeReduced += st.top();

                            st.pop();

                        }

                        if (parsingTable[{st.top(), prevTerminal}] == '<')

                        {

                            break;

                        }

                        else

                        {

                            toBeReduced += st.top();

                            prevTerminal = st.top();

                            st.pop();

                        }

                    }

                    reverse(toBeReduced.begin(), toBeReduced.end());

                    // cout << toBeReduced << endl;

                    productionFound = false;

                    for (auto production : productions)

                    {

                        if (production.second == toBeReduced)

                        {

                            productionFound = true;

                            st.push(production.first);

                            break;

                        }

                    }

                    if (!productionFound)

                    {

                        return false;

                    }

                }

                else

                {

                    return false;

                }

                // display\_stack(st);

            }

        }

    }

    void displayProductions()

    {

        for (auto production : productions)

        {

            cout << production.first << " " << production.second << endl;

        }

    }

    void display\_stack(stack<char> st)

    {

        stack<char> st2;

        while (!st.empty())

        {

            st2.push(st.top());

            st.pop();

        }

        while (!st2.empty())

        {

            cout << st2.top() << " ";

            st2.pop();

        }

    }

    void display\_string(string input, int ptr)

    {

        for (int i = ptr; i < input.size(); i++)

        {

            cout << input[i];

        }

        cout << endl;

    }

};

int main()

{

    ShiftReduceParser parser;

    parser.setProductions();

    parser.displayProductions();

    string input;

    cout << "Enter the string to parse it: ";

    cin >> input;

    if (parser.parse(input))

    {

        cout << "Accepted";

    }

    else

    {

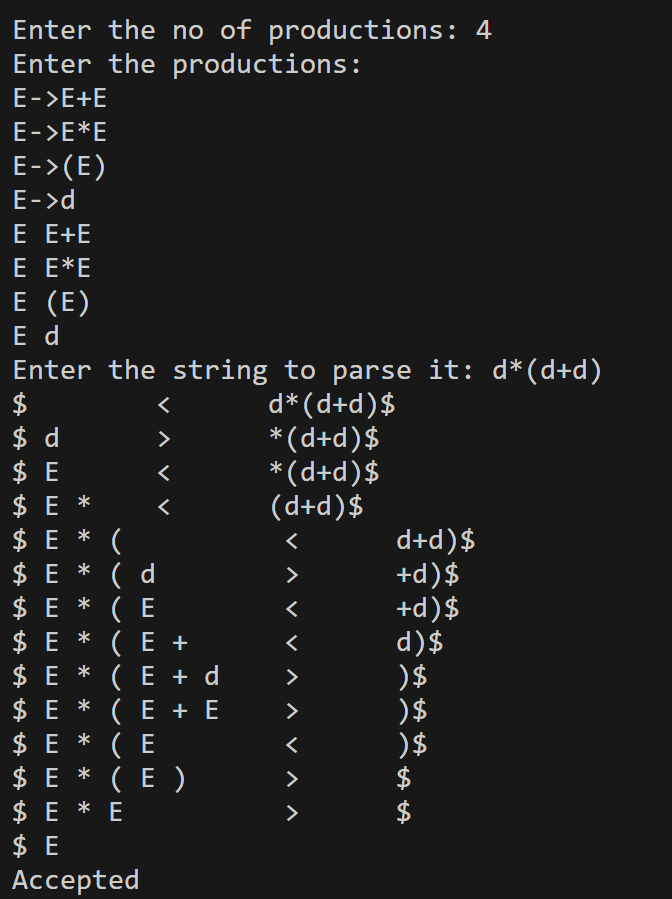
        cout << "Not Accepted";

    }

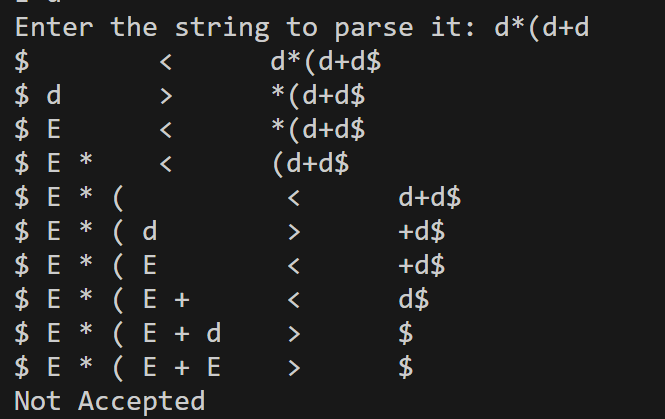
}

**Output:**

**For valid string**



**For Invalid string**

****