**CS 6109**

**COMPILER DESIGN**

**LAB – 6**

**26-10-2021**

**DONE BY**

**SUDARSAN KUMAR N**

**2019503564**

1. **Write a C program and construct SLR parsing table for the grammar and find moves made by predictive parser on input id + id \* id and find FIRST and FELLOW.**

**E --> E + T**

**E --> T**

**T --> T \* F**

**T --> F**

**F --> (E) | id**

**CODE**

#include <bits/stdc++.h>

#define error(x) cerr << #x << " = " << x << '\n'

using namespace std;

set<char> ss;

map<char, vector<vector<char>>> mp;

bool dfs(char i, char org, char last, map<char, vector<vector<char>>> &mp)

{

    bool rtake = false;

    for (auto r : mp[i])

    {

        bool take = true;

        for (auto s : r)

        {

            if (s == i)

                break;

            if (!take)

                break;

            if (!(s >= 'A' && s <= 'Z') && s != 'e')

            {

                ss.insert(s);

                break;

            }

            else if (s == 'e')

            {

                if (org == i || i == last)

                    ss.insert(s);

                rtake = true;

                break;

            }

            else

            {

                take = dfs(s, org, r[r.size() - 1], mp);

                rtake |= take;

            }

        }

    }

    return rtake;

}

map<int, map<char, set<pair<deque<char>, deque<char>>>>> f;

map<int, vector<pair<int, char>>> g;

int num = -1;

void dfs2(char c, char way, int last, pair<deque<char>, deque<char>> curr)

{

    map<char, set<pair<deque<char>, deque<char>>>> mp2;

    int rep = -2;

    if (last != -1)

    {

        for (auto q : g[last])

        {

            if (q.second == way)

            {

                rep = q.first;

                mp2 = f[q.first];

            }

        }

    }

    mp2[c].insert(curr);

    int count = 10;

    while (count--)

    {

        for (auto q : mp2)

        {

            for (auto r : q.second)

            {

                if (!r.second.empty())

                {

                    if (r.second.front() >= 'A' && r.second.front() <= 'Z')

                    {

                        for (auto s : mp[r.second.front()])

                        {

                            deque<char> st, emp;

                            for (auto t : s)

                                st.push\_back(t);

                            mp2[r.second.front()].insert({emp, st});

                        }

                    }

                }

            }

        }

    }

    for (auto q : f)

    {

        if (q.second == mp2)

        {

            g[last].push\_back({q.first, way});

            return;

        }

    }

    if (rep == -2)

    {

        f[++num] = mp2;

        if (last != -1)

            g[last].push\_back({num, way});

    }

    else

    {

        f[rep] = mp2;

    }

    int cc = num;

    for (auto q : mp2)

    {

        for (auto r : q.second)

        {

            if (!r.second.empty())

            {

                r.first.push\_back(r.second.front());

                r.second.pop\_front();

                dfs2(q.first, r.first.back(), cc, r);

            }

        }

    }

}

int main()

{

    int i, j;

    ifstream fin("grammar.txt");

    string num;

    vector<int> fs;

    vector<vector<int>> a;

    char start;

    bool flag = 0;

    cout << "Grammar: " << '\n';

    while (getline(fin, num))

    {

        if (flag == 0)

            start = num[0], flag = 1;

        cout << num << '\n';

        vector<char> temp;

        char s = num[0];

        for (i = 3; i < num.size(); i++)

        {

            if (num[i] == '|')

            {

                mp[s].push\_back(temp);

                temp.clear();

            }

            else

                temp.push\_back(num[i]);

        }

        mp[s].push\_back(temp);

    }

    map<char, set<char>> fmp;

    for (auto q : mp)

    {

        ss.clear();

        dfs(q.first, q.first, q.first, mp);

        for (auto g : ss)

            fmp[q.first].insert(g);

    }

    cout << '\n';

    cout << "FIRST: " << '\n';

    for (auto q : fmp)

    {

        string ans = "";

        ans += q.first;

        ans += " = {";

        for (char r : q.second)

        {

            ans += r;

            ans += ',';

        }

        ans.pop\_back();

        ans += "}";

        cout << ans << '\n';

    }

    map<char, set<char>> gmp;

    gmp[start].insert('$');

    int count = 10;

    while (count--)

    {

        for (auto q : mp)

        {

            for (auto r : q.second)

            {

                for (i = 0; i < r.size() - 1; i++)

                {

                    if (r[i] >= 'A' && r[i] <= 'Z')

                    {

                        if (!(r[i + 1] >= 'A' && r[i + 1] <= 'Z'))

                        {

                            gmp[r[i]].insert(r[i + 1]);

                        }

                        else

                        {

                            char temp = r[i + 1];

                            int j = i + 1;

                            while (temp >= 'A' && temp <= 'Z')

                            {

                                if (\*fmp[temp].begin() == 'e')

                                {

                                    for (auto g : fmp[temp])

                                    {

                                        if (g == 'e')

                                            continue;

                                        gmp[r[i]].insert(g);

                                    }

                                    j++;

                                    if (j < r.size())

                                    {

                                        temp = r[j];

                                        if (!(temp >= 'A' && temp <= 'Z'))

                                        {

                                            gmp[r[i]].insert(temp);

                                            break;

                                        }

                                    }

                                    else

                                    {

                                        for (auto g : gmp[q.first])

                                            gmp[r[i]].insert(g);

                                        break;

                                    }

                                }

                                else

                                {

                                    for (auto g : fmp[temp])

                                    {

                                        gmp[r[i]].insert(g);

                                    }

                                    break;

                                }

                            }

                        }

                    }

                }

                if (r[r.size() - 1] >= 'A' && r[r.size() - 1] <= 'Z')

                {

                    for (auto g : gmp[q.first])

                        gmp[r[i]].insert(g);

                }

            }

        }

    }

    cout << '\n';

    cout << "FOLLOW: " << '\n';

    for (auto q : gmp)

    {

        string ans = "";

        ans += q.first;

        ans += " = {";

        for (char r : q.second)

        {

            ans += r;

            ans += ',';

        }

        ans.pop\_back();

        ans += "}";

        cout << ans << '\n';

    }

    string temp = "";

    temp += '.';

    temp += start;

    deque<char> emp;

    deque<char> st;

    st.push\_back(start);

    dfs2('!', 'k', -1, {emp, st});

    cout << "\nProductions: " << '\n';

    int cc = 1;

    set<char> action, go;

    map<pair<char, deque<char>>, int> pos;

    for (auto q : mp)

    {

        go.insert(q.first);

        for (auto r : q.second)

        {

            cout << "r" << cc << ": ";

            string ans = "";

            ans += q.first;

            ans += "->";

            deque<char> temp;

            for (auto s : r)

                ans += s, temp.push\_back(s);

            pos[{q.first, temp}] = cc;

            for (auto s : r)

            {

                if (s >= 'A' && s <= 'Z')

                    go.insert(s);

                else

                    action.insert(s);

            }

            cout << ans << '\n';

            cc++;

        }

    }

    cout << "\nGraph: " << '\n';

    for (auto mp2 : f)

    {

        cout << '\n';

        cout << "I";

        cout << mp2.first << ": \n";

        for (auto q : mp2.second)

        {

            string ans = "";

            ans += q.first;

            ans += "->";

            for (auto r : q.second)

            {

                for (auto t : r.first)

                    ans += t;

                ans += '.';

                for (auto t : r.second)

                    ans += t;

                ans += '|';

            }

            ans.pop\_back();

            for (auto tt : ans)

            {

                if (tt == '!')

                    cout << start << '\'';

                else

                    cout << tt;

            }

            cout << '\n';

        }

    }

    cout << '\n';

    cout << "Edges: " << '\n';

    for (auto q : g)

    {

        for (auto r : q.second)

        {

            cout << "I" << q.first << " -> " << r.second << " -> "

                 << "I" << r.first << "\n";

        }

    }

    action.insert('$');

    cout << "\nParsing Table:" << '\n';

    cout << "St.\t\tAction & Goto" << '\n';

    int tot = f.size();

    cout << "  \t";

    for (auto q : action)

        cout << q << '\t';

    for (auto q : go)

        cout << q << '\t';

    cout << '\n';

    for (i = 0; i < tot; i++)

    {

        cout << "I" << i << '\t';

        for (auto q : action)

        {

            if (g.count(i))

            {

                int flag = 0;

                for (auto r : g[i])

                {

                    if (r.second == q)

                    {

                        flag = 1;

                        cout << "S" << r.first << "\t";

                        break;

                    }

                }

                if (!flag)

                    cout << "-" << '\t';

            }

            else

            {

                int flag = 0;

                for (auto r : f[i])

                {

                    if (r.first == '!')

                    {

                        if (q == '$')

                        {

                            cout << "AC\t";

                            flag = 1;

                        }

                        else

                            cout << "-\t";

                    }

                }

                if (!flag)

                {

                    for (auto r : f[i])

                    {

                        char ccc = r.first;

                        deque<char> chk = (\*r.second.begin()).first;

                        int cou = 1;

                        for (auto r : gmp[ccc])

                        {

                            if (q == r)

                            {

                                cout << "r" << pos[{ccc, chk}] << "\t";

                            }

                            cou++;

                        }

                    }

                }

            }

        }

        for (auto q : go)

        {

            if (g.count(i))

            {

                int flag = 0;

                for (auto r : g[i])

                {

                    if (r.second == q)

                    {

                        flag = 1;

                        cout << r.first << "\t";

                        break;

                    }

                }

                if (!flag)

                    cout << "-" << '\t';

            }

            else

            {

                cout << "-" << '\t';

            }

        }

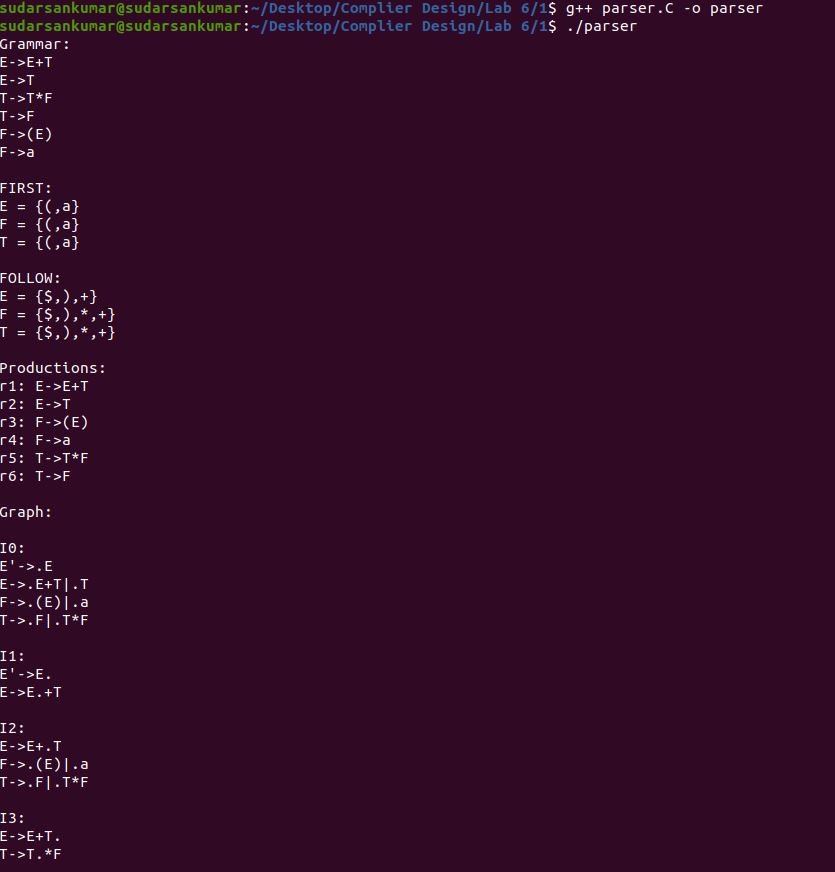
        cout << '\n';

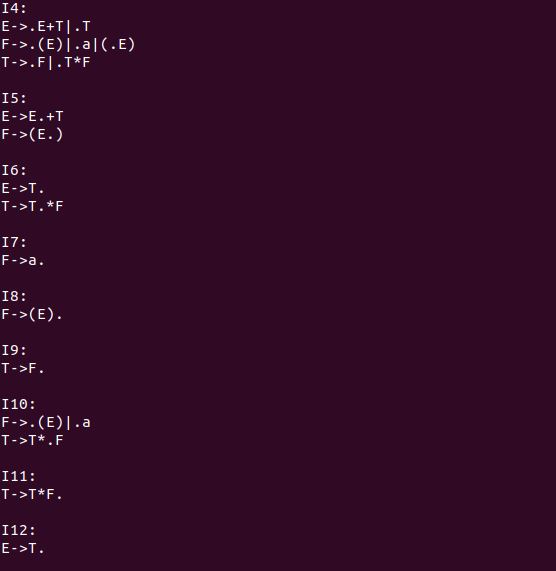
    }

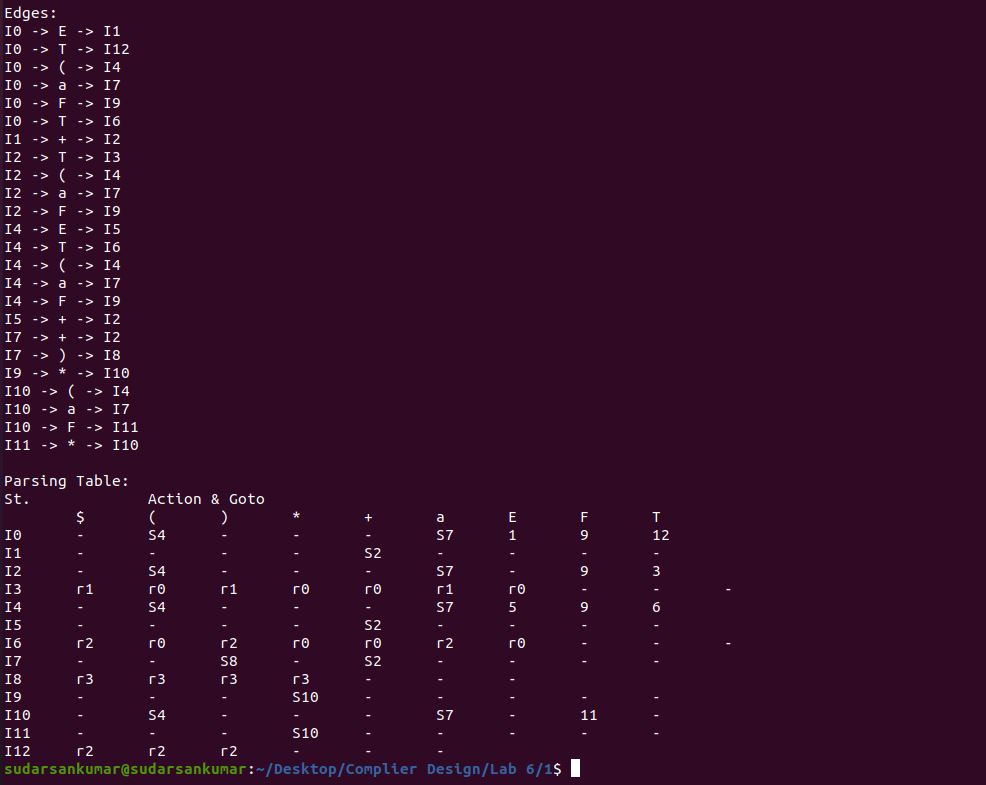
    return 0;

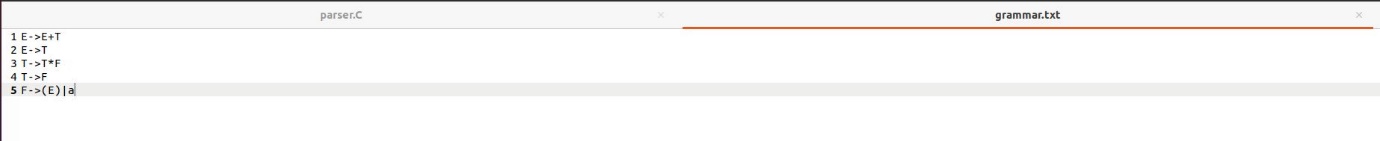
}

**OUTPUT**









1. **Write a C program and construct Operator Precedence parsing table for the grammar and find moves made by predictive parser on input id + id \* id and find FIRST and FELLOW.**

**§E → EAE | id**

**§A → + | x**

**CODE**

#include<stdio.h>

#include<string.h>

char \*input;

int i=0;

char lasthandle[6],stack[50],handles[][5]={")E(","E\*E","E+E","i","E^E"};

int top=0,l;

char prec[9][9]={'>', '>','<','<','<','<','<','>','>',

          '>', '>','<','<','<','<','<','>','>',

          '>', '>','>','>','<','<','<','>','>',

          '>', '>','>','>','<','<','<','>','>',

          '>', '>','>','>','<','<','<','>','>',

          '>', '>','>','>','>','e','e','>','>',

          '<', '<','<','<','<','<','<','>','e',

          '>', '>','>','>','>','e','e','>','>',

          '<', '<','<','<','<','<','<','<','>',};

int getindex(char c)

{

    switch(c)

        {

            case '+':return 0;

            case '-':return 1;

            case '\*':return 2;

            case '/':return 3;

            case '^':return 4;

            case 'i':return 5;

            case '(':return 6;

            case ')':return 7;

            case '$':return 8;

        }

}

int shift()

{

    stack[++top]=\*(input+i++);

    stack[top+1]='\0';

}

int reduce()

{

    int i,len,found,t;

    for(i=0;i<5;i++)

    {

        len=strlen(handles[i]);

        if(stack[top]==handles[i][0]&&top+1>=len)

        {

            found=1;

            for(t=0;t<len;t++)

            {

                if(stack[top-t]!=handles[i][t])

                {

                    found=0;

                    break;

                }

            }

            if(found==1)

            {

                stack[top-t+1]='E';

                top=top-t+1;

                strcpy(lasthandle,handles[i]);

                stack[top+1]='\0';

                return 1;

            }

        }

    }

    return 0;

}

void dispstack()

{

    int j;

    for(j=0;j<=top;j++)

    {

        printf("%c",stack[j]);

    }

}

void dispinput()

{

    int j;

    for(j=i;j<l;j++)

    {

        printf("%c",\*(input+j));

    }

}

void main()

{

    int j;

    input=(char\*)malloc(50\*sizeof(char));

    printf("\nEnter the string\n");

    scanf("%s",input);

    input=strcat(input,"$");

    l=strlen(input);

    strcpy(stack,"$");

    printf("\nSTACK\tINPUT\tACTION");

    while(i<=l)

    {

        shift();

        printf("\n");

        dispstack();

        printf("\t");

        dispinput();

        printf("\tShift");

        if(prec[getindex(stack[top])][getindex(input[i])]=='>')

        {

            while(reduce())

            {

                printf("\n");

                dispstack();

                printf("\t");

                dispinput();

                printf("\tReduced: E->%s",lasthandle);

            }

        }

    }

    if(strcmp(stack,"$E$")==0)

    {

        printf("\n\n\nThe String %s is Accepted\n\n",input);

    }

    else

    {

        printf("\n\n\nThe String %s is Not Accepted\n\n",input);

    }

}

**OUTPUT**

