【Experiment name】

LL (1) Conversion and judgment of grammar

【Purpose】

Input: Arbitrary context-free grammar.

Output: (1) Whether it is LL(1) grammar;

(2) If it is **an LL(1) grammar, output the select set** of each production ;

(3) If it is not **an** LL(1) grammar, check whether it contains left common factors or contains left recursion, and use the corresponding algorithm to change the non- LL(1) grammar into **an** LL(1) grammar, and output each item in the new grammar **The selection set** of productions .

【Experimental principle】

1. Suppose the grammar G[S]=(VN, VT, P, S), then the initial character set is:

FIRST(α)={a | α *aβ, a V ∈ T, α, β V ∈ *}. If α * ε, ε ∈ FIRST(α).

It can be seen from the definition that FIRST(α) refers to all symbol strings that can be deduced by symbol string α.

A collection of terminal symbols at the beginning of a string. So the FIRST set is also called the first symbol set.

Let α=x1x2...xn, FIRST (α) can be obtained by the following method:

Let FIRST(α)=Φ, i=1;

(1) If xi ∈ VT, then xi ∈ FIRST(α);

(2) If xi ∈ VN;

①If ε FIRST(xi), then FIRST(xi) ∈ FIRST(α);

②If ε ∈ FIRST(xi), then FIRST(xi)－{ε} FIRST ∈ (α);

(3) i=i+1, repeat (1), (2) until xi ∈ VT, (i=2, 3,..., n) or xi ∈ VN and if ε FIRST(xi) or i>n.

2. When there is an ε production in a grammar, for example, there is A → ε, only by knowing which symbols can legally appear after the non-terminal A, can we know whether to choose the A → ε production. The set of symbols that legally appear after the nonterminal A is called the FOLLOW set. Below we give the definition of the FOLLOW set of the grammar .

Suppose grammar G[S]=(V N, V T, P, S), then

FOLLOW(A) = {a | S ⇒... Aa ..., a ∈ V T}.

If S ⇒*...A, # FOLLOW ∈ (A).

It can be seen from the definition that FOLLOW(A) refers to the set of terminal symbols that immediately follow the non-terminal symbol A in all sentence patterns of the grammar G[S].

The FOLLOW set can be obtained as follows:

(1) For the start symbol S of the grammar G[S], there is # FOLLOW ∈ (S);

(2) If there are rules of the form B→xAy in the grammar G[S], where x, y V ∈ *, then FIRST(y)－{ε}∈FOLLOW(A);

(3) If there are rules of the form B→xA in the grammar G[S], or rules of the form B→xAy and ε∈FIRST(y), where x, y V ∈ *, then FOLLOW(B) ∈FOLLOW (A);

3. Calculate the SELECT(A→α) set of each production A→α

Computes SELECT(A→α) by definition:

(1) If α can never be derived from ε, then SELECT(A→α)=FIRST(α)

(2) If α may be derived from ε, then SELECT(A→α)=(FIRST(α)-{ε}) ∪FOLLOW(A)

4. A context-free grammar G is an LL(1) grammar if and only if any two different rules A→α｜β for each nonterminal A in G satisfy SELECT(A→α)∩SELECT(A→β)=Φ, where at most one of α and β can deduce ε string.

【Experimental content】

code:

```
#include <iostream>
#include <string>
#include <algorithm>
#include <fstream>
#include <iomanip>
#include <vector>
using namespace std;

#define MAXS 50
int NONE[MAXS] = { 0 };
int visit[MAXS] = { 0 };
string strings;// production
string Vn;// non-terminal character
string Vt;// terminator
string first[MAXS];// used to store the first set of each terminal
string First[MAXS];// used to store the First set of each non-terminal
string Follow[MAXS]; // Follow set used to store each non-terminal
int N = 0;//Number of productions
char initialstate;
int character = 0;

struct LRS
{
	string left;
	string right;
	string select;
};
class First_Follow_Select_LL
{
public:
	void VNVT(LRS* p);
	string Letter_First(LRS* p, char ch);
	string Letter_Follow(LRS* p, char ch);
```

```
        void Letter_Select(LRS* p);
        void Open_File(int& n, LRS* p);
        void Display(LRS* p);
        int Judge_LL(LRS* p);
        void Select_Show(LRS* p);
        void Change_LL(LRS* p, int info);
        void Combine(LRS* p);
        void Extract(LRS* p);
        int Easy_Judge(LRS* p);
};
//求 VN  和 VT
void First_Follow_Select_LL::VNVT(LRS* p)
{
        int i, j;
        Vn.clear();
        Vt.clear();
        for (i = 0; i < MAXS; i++)
        {
                first[i].clear();
                First[i].clear();
                Follow[i].clear();
        }
        for (i = 0; i < N; i++)
        {
                if ((p[i].left[0] >= 'A' && p[i].left[0] <= 'Z'))
                        if (Vn.find(p[i].left[0]) == string::npos)
                                Vn += p[i].left[0];
                for (j = 0; j < p[i].right.size(); j++)
                {
                        if (p[i].right[j] >= 'A' && p[i].right[j] <= 'Z')
                        {
                                if (Vn.find(p[i].right[j]) == string::npos)
                                        Vn += p[i].right[j];
                        }
                        else
                        {
                                if (Vt.find(p[i].right[j]) == string::npos)
                                        Vt += p[i].right[j];
                        }

                }
        }
}
//对每个文法符号求 first  集
string First_Follow_Select_LL::Letter_First(LRS* p, char ch)
{
        int t;

        if (Vt.find(ch) != string::npos)
        {
                first[Vt.find(ch)] = ch;
                return first[Vt.find(ch)];
        }
        if (Vn.find(ch) != string::npos)
        {
                visit[Vn.find(ch)]++;
                if (visit[Vn.find(ch)] == 2)
                {
                        visit[Vn.find(ch)] = 1;
                        return First[Vn.find(ch)];
                }
                for (int i = 0; i < N; i++)
                {
                        if (p[i].left[0] == ch)
                        {
                                if (Vt.find(p[i].right[0]) != string::npos)
                                {
                                        if (First[Vn.find(ch)].find(p[i].right[0]) == string::npos)
                                        {
```

```
                First[Vn.find(ch)] += p[i].right[0];
        }
}
if (p[i].right[0] == '*')
{
        if (First[Vn.find(ch)].find('*') == string::npos)
        {
                First[Vn.find(ch)] += '*';
        }
}
if (Vn.find(p[i].right[0]) != string::npos)
{
        if (p[i].right.size() == 1)
        {
                string ff;
                if (First[Vn.find(p[i].right[0])] != "")
                {
                        ff = First[Vn.find(p[i].right[0])];
                }
                else
                {
                        ff = Letter_First(p, p[i].right[0]);
                        visit[Vn.find(p[i].right[0])]++;
                }
                for (int ii = 0; ii < ff.size(); ii++)
                {
                        if (First[Vn.find(ch)].find(ff[ii]) == string::npos)
                        {
                                First[Vn.find(ch)] += ff[ii];
                        }
                }
        }
        else
        {

                for (int j = 0; j < p[i].right.size(); j++)
                {
                        if (p[i].right[0] == p[i].left[0])
                                break;
                        string TT;
                        if (First[Vn.find(p[i].right[j])] != "")
                        {
                                TT = First[Vn.find(p[i].right[j])];
                        }
                        else
                        {
                                TT = Letter_First(p, p[i].right[j]);
                                visit[Vn.find(p[i].right[j])]++;
                        }

                        if (TT.find('*') != string::npos && (j + 1) < p[i].right.size())
                        {
                                sort(TT.begin(), TT.end());
                                string tt;
                                for (int t = 1; t < TT.size(); t++)
                                {
                                        tt += TT[t];
                                }
                                TT = tt;
                                tt = "";
                                for (t = 0; t < TT.size(); t++)
                                {
                                        if (First[Vn.find(ch)].find(TT[t]) == string::npos)
                                        {
                                                First[Vn.find(ch)] += TT[t];
                                        }
                                }
                        }
                        else
```

```
                                        {
                                                for (t = 0; t < TT.size(); t++)
                                                {
                                                        if (First[Vn.find(ch)].find(TT[t]) == string::npos)
                                                        {
                                                                First[Vn.find(ch)] += TT[t];
                                                        }
                                                }
                                                break;
                                        }
                                }
                        }
                }
        }
        return First[Vn.find(ch)];
    }
}
//求每个非终结符的 Follow 集
string First_Follow_Select_LL::Letter_Follow(LRS* p, char ch)
{
    int t, k;
    NONE[Vn.find(ch)]++;
    if (NONE[Vn.find(ch)] == 2)
    {
        NONE[Vn.find(ch)] = 1;
        return Follow[Vn.find(ch)];
    }
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < p[i].right.size(); j++)
        {
            if (p[i].right[j] == ch)
            {
                if (j + 1 == p[i].right.size())
                {
                    string gg;
                    gg = Letter_Follow(p, p[i].left[0]);
                    NONE[Vn.find(p[i].left[0])] = 1;
                    for (int k = 0; k < gg.size(); k++)
                    {
                        if (Follow[Vn.find(ch)].find(gg[k]) == string::npos)
                        {
                            Follow[Vn.find(ch)] += gg[k];
                        }
                    }
                }
                else
                {
                    string FF;
                    for (int jj = j + 1; jj < p[i].right.length(); jj++)
                    {
                        string TT;
                        TT = Letter_First(p, p[i].right[jj]);
                        if ((TT.find('*') != string::npos) && (jj + 1) < p[i].right.size())
                        {
                            sort(TT.begin(), TT.end());
                            string tt;
                            for (int t = 1; t < TT.size(); t++)
                            {
                                tt += TT[t];
                            }
                            TT = tt;
                            tt = "";
                            for (t = 0; t < TT.size(); t++)
                            {
                                if (FF.find(TT[t]) == string::npos)
                                    //&& TT[t] != '*'
                                {
```

```
                                        FF += TT[t];
                                    }
                                }
                            }
                            else
                            {
                                for (t = 0; t < TT.size(); t++)
                                {
                                    if (FF.find(TT[t]) == string::npos)
                                    {
                                        FF += TT[t];
                                    }
                                }
                                break;
                            }
                        }
                        if (FF.find('*') == string::npos)
                        {
                            for (k = 0; k < FF.size(); k++)
                            {
                                if (Follow[Vn.find(ch)].find(FF[k]) == string::npos)
                                {
                                    Follow[Vn.find(ch)] += FF[k];
                                }
                            }
                        }
                        else
                        {
                            for (k = 0; k < FF.size(); k++)
                            {
                                if ((Follow[Vn.find(ch)].find(FF[k]) == string::npos) && FF[k] != '*')
                                {
                                    Follow[Vn.find(ch)] += FF[k];
                                }
                            }
                            string dd;
                            dd = Letter_Follow(p, p[i].left[0]);
                            NONE[Vn.find(p[i].left[0])] = 1;
                            for (k = 0; k < dd.size(); k++)
                            {
                                if (Follow[Vn.find(ch)].find(dd[k]) == string::npos)
                                {
                                    Follow[Vn.find(ch)] += dd[k];
                                }
                            }
                        }
                    }
                }
            }
        }
        return Follow[Vn.find(ch)];
}
//Open the file and display the read grammar
void First_Follow_Select_LL::Open_File(int& n, LRS* p)
{
        string temp;
        int i;

        //Select data1: LL(1) data2: left recursive data3: left common factor
        ifstream myfile("f:\data1.txt");
        //ifstream myfile("f:\data2.txt");
        //ifstream myfile("f:\data3.txt");

        if (!myfile)
        {
                cout << "The file you want to open does not exist" << endl;
        }
        getline(myfile, temp);
        initialstate = temp[0];
```

```
	temp. clear();
	while (!myfile. eof())
	{
		getline(myfile, temp);
		p[n].left = temp.substr(0, 1);
		p[n].right = temp.substr(3, temp.size() - 3);
		n++;
		temp. clear();
	}
	myfile. close();
	cout << "Grammar given in the file:" << endl;
	for (i = 0; i < N; i++)
	{
		cout << i + 1 << ": ";
		cout << p[i].left << "->" << p[i].right << endl;
	}
}
//求每一个产生式的 select 集
void First_Follow_Select_LL::Letter_Select(LRS* p)
{
	int i, j, k;
	for (i = 0; i < N; i++)
	{
		for (j = 0; j < p[i].right.size(); j++)
		{
			if (Vt.find(p[i].right[0]) != string::npos)
			{
				if (p[i].right[0] != '*')
					p[i].select = p[i].right[0];
				else
					p[i].select = Follow[Vn.find(p[i].left[0])];
				break;
			}
			else if (Vn.find(p[i].right[j]) != string::npos)
			{
				if (First[Vn.find(p[i].right[j])].find("*") == string::npos)
				{
					p[i].select += First[Vn.find(p[i].right[j])];
					break;
				}
				else
				{
					for (k = 0; k < First[Vn.find(p[i].right[j])].size(); k++)
					{
						if (First[Vn.find(p[i].right[j])][k] != '*' && p[i].select.find(First[Vn.find(p[i].right[j])][k])
== string::npos)
							p[i].select += First[Vn.find(p[i].right[j])][k];
					}
					if (j + 1 == p[i].right.size())
						for (k = 0; k < Follow[Vn.find(p[i].left[0])].size(); k++)
						{
							if (p[i].select.find(Follow[Vn.find(p[i].left[0])][k]) == string::npos)
								p[i].select += Follow[Vn.find(p[i].left[0])][k];
						}
				}
			}
		}
	}
}
//将 select 集显示出来
void First_Follow_Select_LL::Select_Show(LRS* p)
{
	int i, j;
	string temp = "{";
	string temp1;
	for (i = 0; i < N; i++)
	{
		if (p[i].select.size() != 0)
		{
```

```
                for (j = 0; j + 1 < p[i].select.size(); j++)
                {
                        temp += p[i].select[j];
                        temp.append(",");
                }
                temp += p[i].select[p[i].select.size() - 1];
                temp.append("}");
                temp1 = p[i].left + "->" + p[i].right;
                cout <<"SELECT(" << temp1 << ")\t=" << temp << endl;
                temp = "{";
                temp1 = "";
        }
        else
        {
                temp1 = p[i].left + "->" + p[i].right;
                cout << "SELECT(" << temp1 << ")" << "does not exist" << endl;
                temp1 = "";
        }
    }
    cout << "----------------------------------------" << endl;
}
//display the result
void First_Follow_Select_LL::Display(LRS* p)
{
    int i, j, k;
    int tab = 17;
    cout << endl << "Result: " << endl;
    cout << "----------------------------------------" << endl;
    cout << "FIRST 集" << setw(tab) << "FOLLOW 集" << endl;
    Follow[Vn.find(initialstate)] += '*';
    for (i = 0; i < Vn.size(); i++)
    {
        cout << "FIRST(" << Vn[i] << ")=";
        string temp1, temp2, temp3 = "{";
        temp1 = Letter_First(p, Vn[i]);
        if (temp1 != "")
        {
                for (j = 0; j + 1 < temp1.size(); j++)
                {
                        temp3 += temp1[j];
                        temp3.append(",");
                }
                temp3 += temp1[temp1.size() - 1];
                temp3.append("}");
                cout << temp3 ;
                temp3 = "{";
        }
        else
        {
                cout <<   "不存在！ ";
        }
        cout   << "\tFOLLOW(" << Vn[i] << ")=";
        temp2 = Letter_Follow(p, Vn[i]);
        for (k = 0; k < Vn.size(); k++)
        {
                NONE[k] = 0;
                visit[k] = 0;
        }

        for (k = 0; k + 1 < temp2.size(); k++)
        {
                temp3 += temp2[k];
                temp3.append(",");
        }
        if (temp2.size() != 0)
        {
                temp3 += temp2[temp2.size() - 1];
                temp3.append("}");
        }
```

```
            else
            {
                temp3 += "不存在}";
            }
            cout << temp3 << endl;
        }
        cout << "----------------------------------------" << endl;
}
//判断是不是 LL（1）文法
int First_Follow_Select_LL::Judge_LL(LRS* p)
{
        int i, j, k;
        string temp1 = "";
        for (i = 0; i < Vn.size(); i++)
        {
            for (j = 0; j < N; j++)
            {
                if (p[j].left[0] == Vn[i])
                {
                    if (temp1 == "")
                        temp1 = p[j].select;
                    else
                    {
                        for (k = 0; k < temp1. size(); k++)
                        {
                            if (p[j].select.find(temp1[k]) != string::npos)
                            {
                                cout << "This grammar is not an LL(1) grammar!" << endl;
                                return 0;
                            }
                        }
                    }
                }
            }
            temp1 = "";
        }
        cout << "This grammar is LL(1) grammar!" << endl;
        return 1;
}
//Reintegrate the storage of the grammar, remove empty strings, remove useless productions, remove the same productions converted by different conversions, and set select to empty
void First_Follow_Select_LL::Combine(LRS* p)
{
        int i, j;
        string temp;
        temp += initialstate;
        for (i = 0; i < N; i++)
        {
            for (j = 0; j < p[i].right.size(); j++)
            {
                if (temp.find(p[i].right[j]) == string::npos && p[i].right[j] >= 'A' && p[i].right[j] <= ' Z')
                {
                    temp += p[i].right[j];
                }
            }
        }
        for (i = 0; i < N; i++)
        {
            if (temp.find(p[i].left[0]) == string::npos)
            {
                p[i].left = "";
                p[i].right = "";
            }
        }
        temp.clear();
        temp += initialstate;
        for (i = 0; i < N; i++)
        {
            for (j = 0; j < p[i].right.size(); j++)
```

```
            {
                if (temp.find(p[i].right[j]) == string::npos && p[i].right[j] >= 'A' && p[i].right[j] <= 'Z')
                {
                    temp += p[i].right[j];
                }
            }
        }
        for (i = 0; i < N; i++)
        {
            if (p[i].left != "")
                for (j = i + 1; j < N; j++)
                {
                    if (p[i].left == p[j].left && p[i].right == p[j].right)
                    {
                        p[j].left = "";
                        p[j].right = "";
                    }
                }
        }
        LRS* q = new LRS[MAXS];
        for (i = 0, j = 0; i < N; i++)
        {
            if (p[i].left != "")
            {
                q[j].left = p[i].left;
                q[j].right = p[i].right;
                q[j].select = "";
                j++;
            }
        }
        N = j;
        int count = 0;
        for (j = 0; j < temp.size(); j++)
        {
            for (i = 0; i < N; i++)
            {
                if (q[i].left[0] == temp[j])
                {
                    p[count].left = q[i].left;
                    p[count].right = q[i].right;
                    p[count].select = q[i].select;
                    count++;
                }
            }
        }
        N = count;
        delete[]q;
}
//将非 LL（1）文法转换成 LL（1)文法
void First_Follow_Select_LL::Change_LL(LRS* p, int info)
{
        char q[20];
        int i, j, count = 1, count1 = N, flag = 0, m, x;
        q[0] = p[0].left[0];
        for (i = 1; i < N; i++)
        {
            for (j = 0; j < i; j++)
            {
                if (p[i]. left == p[j]. left)
                    break;
            }
            if (j == i)
                q[count++] = p[i].left[0];
        }
        count--;
        for (i = 0; i < N; i++)// determine whether the first non-terminal symbol has direct left recursion
            if (p[i]. left[0] == q[0] && p[i]. left[0] == p[i]. right[0])
                flag++;
        if (flag != 0)// eliminate the direct left recursion of the first non-terminal
```

```
{
	for (i = 0; i < N; i++)
	{
		if (p[i]. left[0] == q[0])
		{
			while (Vn. find(char(character + 65)) != string::npos)
				character++;
			if (p[i]. left[0] == p[i]. right[0])
			{
				//p[i].left = p[i].left + "'";
				p[i].left = char(character + 65);
				p[i].right = p[i].right.substr(1, p[i].right.length()) + p[i].left;
			}
			else
				//p[i].right = p[i].right + p[i].left + "'";
				p[i].right = p[i].right + char(character + 65);
		}
	}
	//p[count1].left = p[0].left;
	p[count1].left = char(character + 65);
	p[count1++].right = "*";// replace the empty production with *
	character++;
}
// indirect left recursion replacing the first non-terminal
if (info == 1)
{
	for (i = 0; i < count1; i++)
	{
		if (p[i].left[0] == q[0] && p[i].right[0] != p[i].left[0] && p[i].right[0] >= 'A' && p[i].right[0] <= 'Z')
		{
			for (j = 0; j < count1; j++)
			{
				if (p[j].left[0] == p[i].right[0])
				{
					if (p[j].right != "*")
					{
						p[count1].left = p[i].left;
						p[count1].right = p[j].right + p[i].right.substr(1, p[i].right.size() - 1);
						count1++;
					}
					else
					{
						if (p[i].right.size() > 1)
						{
							p[count1].left = p[i].left;
							p[count1].right = p[i].right.substr(1, p[i].right.length());
							count1 = count1 + 1;
						}
						else
						{
							p[count1].left = p[i].left;
							p[count1].right = p[j].right;
							count1 = count1 + 1;
						}
					}
				}
			}
			p[i].left = "";
			p[i].right = "";
		}

	}
}
//消一切左递归
for (m = 0; m <= count; m++)
{
	for (i = 0; i < N; i++)
	{
		if (p[i].left[0] == q[m])
```

```
        {
                for (j = 0; j < count1; j++)
                {
                        for (x = m + 1; x <= count; x++)
                                if (p[j].left[0] == q[x] && p[j].right[0] == q[m])
                                {
                                        if (p[i].right != "*")
                                        {
                                                p[count1].left = p[j].left;
                                                p[count1].right = p[i].right + p[j].right.substr(1, p[j].right.length());
                                                count1 = count1 + 1;
                                        }
                                        else
                                        {
                                                if (p[j].right.size() > 1)
                                                {
                                                        p[count1].left = p[j].left;
                                                        p[count1].right = p[j].right.substr(1, p[j].right.length());
                                                        count1 = count1 + 1;
                                                }
                                                else
                                                {
                                                        p[count1].left = p[j].left;
                                                        p[count1].right = p[i].right;
                                                        count1 = count1 + 1;
                                                }
                                        }
                                }
                }
        }
}
for (j = 0; j < count1; j++)
{
        for (x = m + 1; x <= count; x++)
                if (p[j].right[0] == q[m] && p[j].left[0] == q[x])
                {
                        p[j].right = "";
                        p[j]. left = "";
                }
}
for (x = 0, flag = 0; x < count1; x++)// determine whether there is direct left recursion in the mth non-terminal
{
        if (p[x]. left[0] == q[m] && p[x]. left[0] == p[x]. right[0])
                flag++;         //eliminate direct left recursion
}
if (flag != 0)
{
        for (i = 0; i < count1; i++)
        {
                if (p[i].left[0] == q[m])
                {
                        while (Vn.find(char(character + 65)) != string::npos)
                                character++;
                        if (p[i].left[0] == p[i].right[0])
                        {
                                //p[i].left = p[i].left + "'";
                                p[i].left = char(character + 65);
                                p[i].right = p[i].right.substr(1, p[i].right.length()) + p[i].left;
                                p[count1].left = p[i].left;
                                p[count1].right = "*";//  用*代替空产生式
                        }
                        else
                                //p[i].right = p[i].right + p[i].left + "'";
                                p[i].right = p[i].right + char(character + 65);
                }
        }
        character++;
        count1 = count1 + 1;
}
```

```
        }
        count1;
        //Extract the left common factor
        N = count1;
}
//Determine whether there is a left common factor in the grammar
int First_Follow_Select_LL::Easy_Judge(LRS* p)
{
        int i, j;
        for (i = 0; i < N; i++)
        {
                for (j = i + 1; j < N; j++)
                {
                        if (p[i]. left[0] == p[j]. left[0] && p[i]. right[0] == p[j]. right[0])
                                return 0;
                }
        }
        return 1;
}
//Extract the left common factor
void First_Follow_Select_LL::Extract(LRS* p)
{
        int i, j, k, count = N, l;
        string Vntemp, Vttemp;
        for (i = 0; i < N; i++)
        {
                if (Vntemp.find(p[i].left[0]) == string::npos && p[i].left[0] >= 'A' && p[i].left[0] <= 'Z')
                        Vntemp += p[i].left[0];
                for (j = 0; j < p[i].right.size(); j++)
                        if (Vttemp.find(p[i].right[j]) == string::npos && (p[i].right[j] >= 'a' && p[i].right[j] <= 'z' || p[i].right[j] ==
'*'))
                                Vttemp += p[i].right[j];
        }
        for (i = 0; i < Vntemp.size(); i++)
        {
                for (j = 0; j < N; j++)
                {
                        if (p[j].left[0] == Vntemp[i])
                        {
                                vector<string> temp;
                                temp.push_back(p[j].right);
                                for (k = j + 1; k < N; k++)
                                {
                                        if (p[k].right[0] == p[j].right[0] && p[k].left[0] == Vntemp[i])
                                        {
                                                for (l = 1; l < p[k].right.size();)
                                                {
                                                        if (p[j].right[l] == p[k].right[l])
                                                        {
                                                                l++;
                                                                if (l < p[k].right.size())
                                                                        continue;
                                                                else
                                                                {
                                                                        for (vector<string>::iterator it = temp.begin(); it != temp.end();)
                                                                        {
                                                                                if (*it == p[j].right)
                                                                                        it = temp.erase(it);
                                                                                else
                                                                                        it++;
                                                                        }
                                                                        while (Vntemp.find(char(character + 65)) != string::npos)
                                                                                character++;

                                                                        p[count].left = char(character + 65);
                                                                        if (p[k].right.substr(l, p[k].right.size() - l) != "")
                                                                                p[count].right = p[k].right.substr(l, p[k].right.size() - l);
                                                                        else
                                                                                p[count].right = "*";
```

```
                                        p[k].left = "";
                                        p[k].right = "";
                                        count++;
                                        p[count].left = char(character + 65);
                                        if (p[j].right.substr(l, p[j].right.size() - 1) != "")
                                                p[count].right = p[j].right.substr(l, p[j].right.size() - 1);
                                        else
                                                p[count].right = "*";
                                        count++;
                                        p[j].right = p[j].right.substr(0, l) + char(character + 65);
                                        temp.push_back(p[j].right);
                                        break;
                                }
                        }
                        else
                        {
                                if (l > 1)
                                {
                                        for (vector<string>::iterator it = temp.begin(); it != temp.end();)
                                        {
                                                if (*it == p[j].right)
                                                        it = temp.erase(it);
                                                else
                                                        it++;
                                        }
                                        while (Vntemp.find(char(character + 65)) != string::npos)
                                                character++;

                                        p[count].left = char(character + 65);
                                        if (p[k].right.substr(l, p[k].right.size() - l) != "")
                                                p[count].right = p[k].right.substr(l, p[k].right.size() - l);
                                        else
                                                p[count].right = "*";
                                        p[k].left = "";
                                        p[k].right = "";
                                        count++;
                                        p[count].left = char(character + 65);
                                        if (p[j].right.substr(l, p[j].right.size() - 1) != "")
                                                p[count].right = p[j].right.substr(l, p[j].right.size() - 1);
                                        else
                                                p[count].right = "*";
                                        count++;
                                        p[j].right = p[j].right.substr(0, l) + char(character + 65);
                                        temp.push_back(p[j].right);
                                }
                                else
                                {
                                        temp.push_back(p[k].right);
                                        p[k].left = "";
                                        p[k].right = "";
                                }
                                break;
                        }
                }
                N = count;
        }
}
if (temp.size() > 1)
{
        while (Vntemp.find(char(character + 65)) != string::npos)
                character++;
        p[j].right = p[j].right.substr(0, 1) + char(character + 65);
        for (k = 0; k < temp.size(); k++)
        {
                p[count].left = char(character + 65);
                if (temp.at(k).substr(1, temp.at(k).size() - 1) != "")
                        p[count].right = temp.at(k).substr(1, temp.at(k).size() - 1);
                else
                        p[count].right = "*";
```

```
                                        count++;
                                }
                                N = count;
                        }
                        temp. clear();
                }
        }
    }
}
//main function
int main(int argc, char* argv[])
{
    LRS* p = new LRS[MAXS];
    int flag;
    First_Follow_Select_LL fifo;
    fifo.Open_File(N, p);
    fifo. Combine(p);
    fifo.VNVT(p);//find VN and VT
    fifo.Display(p);//display the result
    fifo.Letter_Select(p);//Find the select set of each production
    fifo.Select_Show(p);//Display the select set
    int iterout = 0;
    while (!fifo.Judge_LL(p))//judging whether it is LL (1) grammar
    {
        cout << "*conversion operation*";
        flag = 1;
        int iterin = 0;
        fifo.Change_LL(p, flag);
        fifo. Combine(p);
        while (!fifo.Easy_Judge(p))
        {
            iterin++;
            fifo. Extract(p);
            fifo. Combine(p);
            if (iterin >= 10)
                break;
        }
        fifo.VNVT(p);
        fifo. Display(p);
        fifo. Letter_Select(p);
        fifo. Select_Show(p);
        iterout++;
        if (iterout >= 10)
        {
            cout << "This grammar cannot be converted to an LL(1) grammar!" << endl;
            break;
        }
    }
    return 0;
}
```

【Experimental Results】

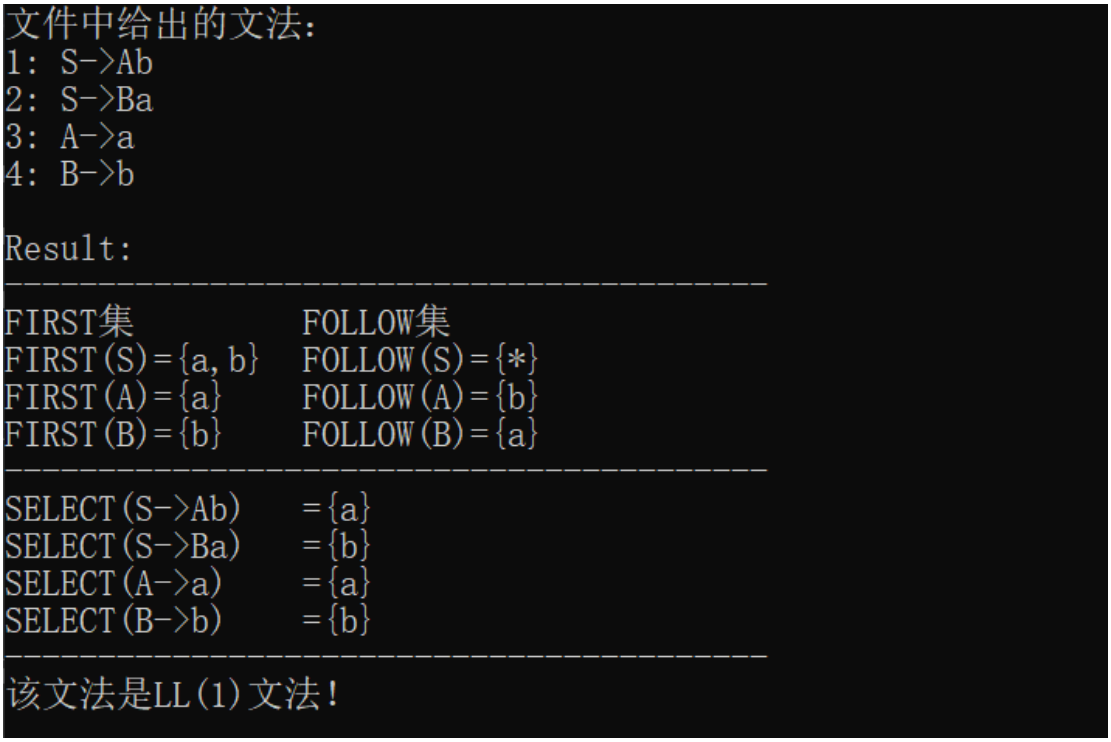

Figure 1: LL (1) grammar

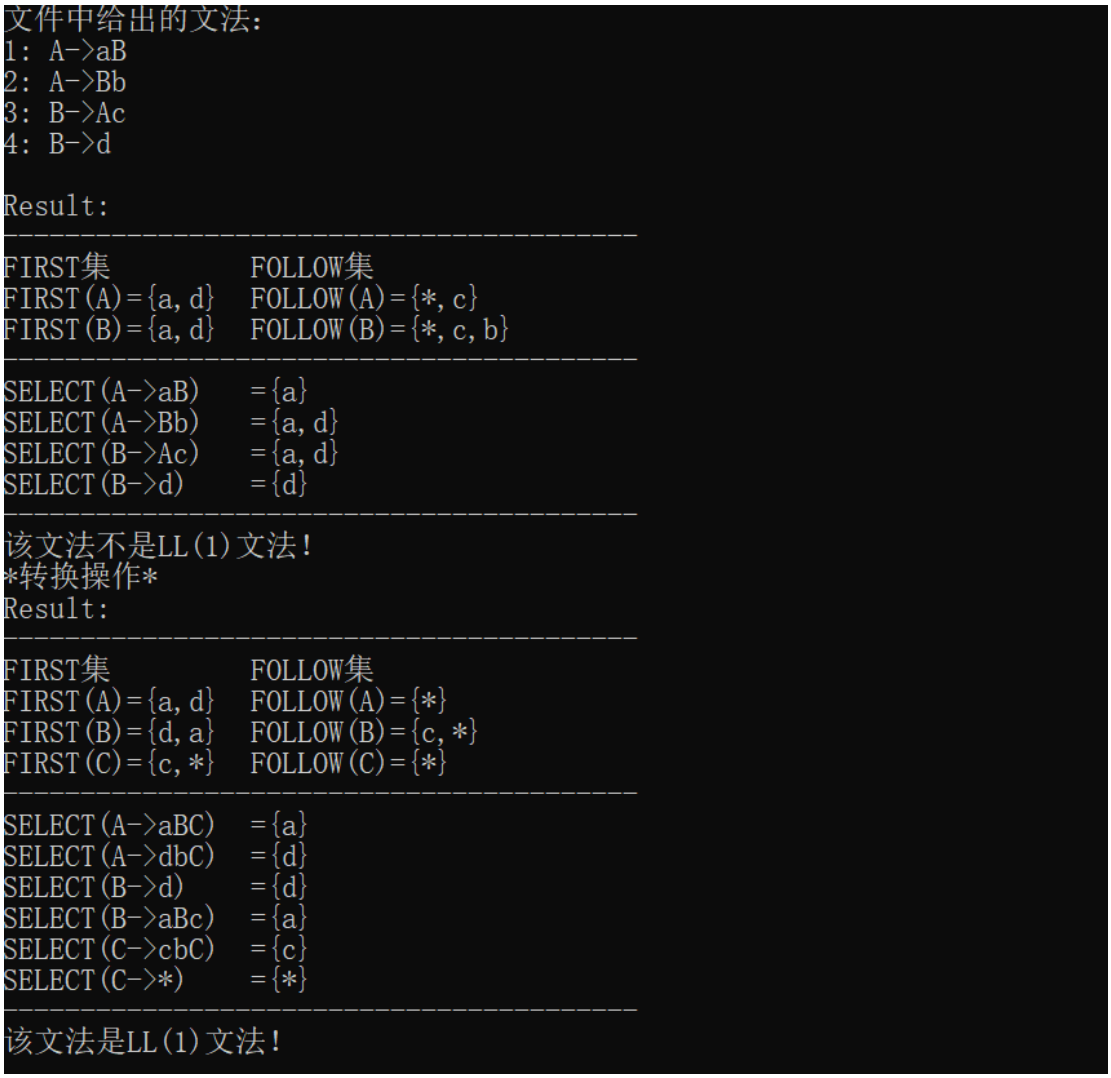

Figure 2: With left recursion

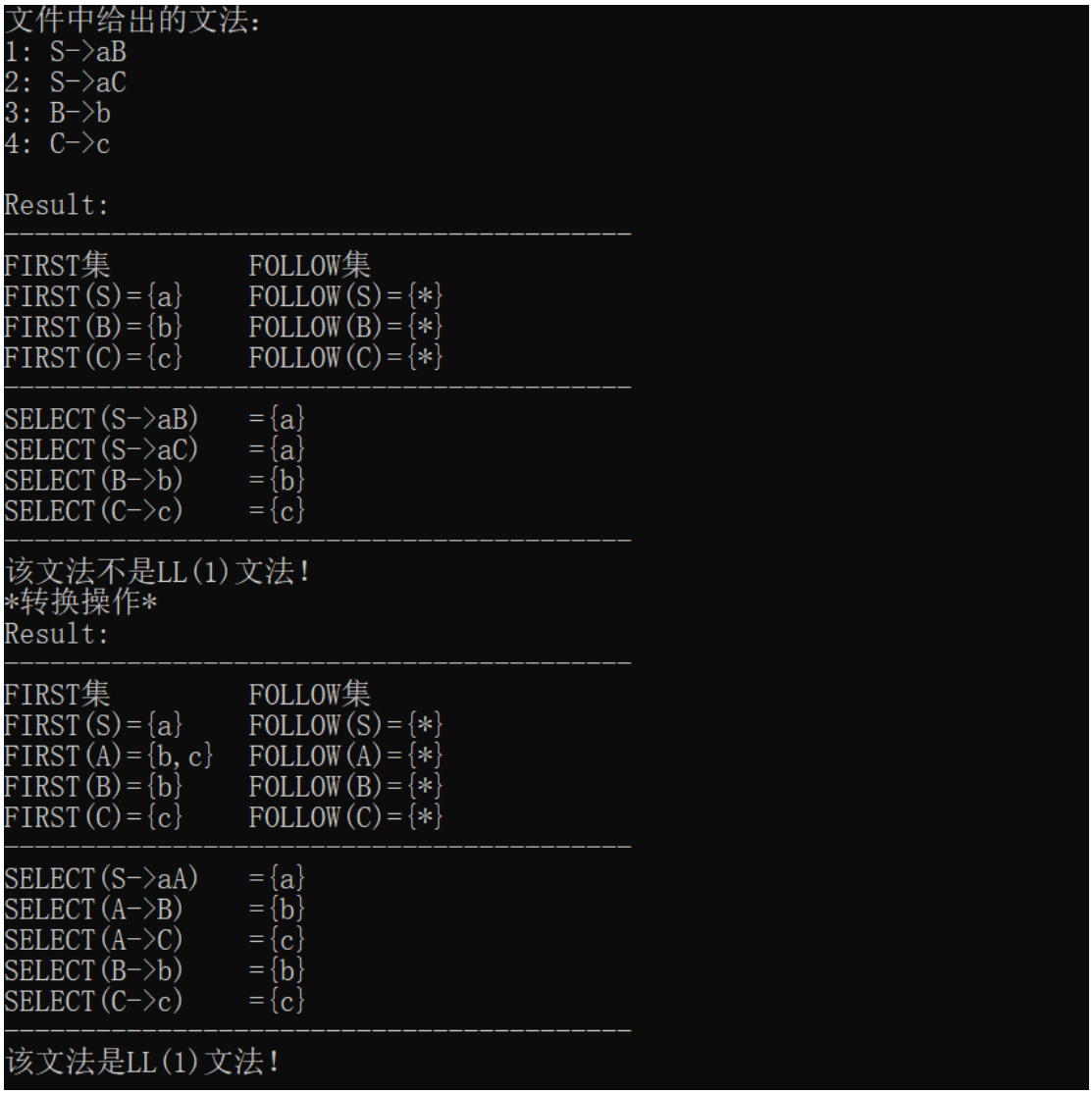

Figure 3: Containing left common factors

【Experiment Summary】

This experiment focuses on the analysis process of LL (1) grammar and the way of judging whether the grammar is LL (1) grammar. This experiment is more difficult. I have tried many times and debugged many times, and I often encounter various difficulties. Finally, through querying references and my own understanding and thinking about the code, I finally completed this experiment. During the experiment, I found that my previous understanding of the chapter

on LL (1) grammar was still partially impenetrable. After experiments and repeated tests, I really understood the contents of each part of this chapter. The experiment is mainly divided into two parts, set generation and grammar conversion, and set generation is divided into three small parts: F IRST , FOLLOW and SELECT. In txt , the first line records the starting point, and the subsequent lines are recorded as output results. When the program judges that it is not an LL(1) grammar, it will convert it so that it conforms to the specification of the LL (1) grammar. After this experiment, I have a deeper understanding of the analysis and judgment methods of LL (1) grammar, which will lay the foundation for future experiments.