北京交通大學

BEIJING JIAOTONG UNIVERSITY

《编译原理》实验报告

实验名称：专题三_LL(1)语法分析设计原理与实现

学　　号：22281188

姓　　名：江家玮

学　　院：计算机科学与技术学院

日　　期：2024 年 12 月 12 日

目录

1. 程序功能描述

以专题 1 词法分析程序的输出为语法分析的输入，实现 LL(1)分析中控制程序（表驱动程序）；完成以下描述赋值语句的 LL(1)文法的 LL(1)分析过程。

重点 LL（1）分析方法和 LL(1)分析器的实现。 用 LL(1)方法判断输入的二元式是否满足文法需求。

2. 主要数据结构描述

2.1 grammar (文法规则)

类型：map<string, vector<string>>

描述：存储文法的产生式，键是非终结符，值是一个包含该非终结符所有可能右部的字符串向量。

例如："S" -> {"V=E"} 表示非终结符 S 有一个产生式 V=E。

2.2 Vt (终结符集合)

类型：set<string>

描述：存储所有的终结符。终结符是文法中无法再分解的符号。

2.3 Vn (非终结符集合)

类型：set<string>

描述：存储所有的非终结符。非终结符是可以被替换成其他符号（包括终结符和非终结符）的符号。

2.4 firsts (FIRST 集合)

类型：map<string, set<string>>

描述：存储每个符号（非终结符）对应的 FIRST 集合。FIRST 集合包含一个符号（非终结符）能够推导出的所有终结符。

键是非终结符或终结符，值是该符号的 FIRST 集合。

2.5 follows (FOLLOW 集合)

类型：map<string, set<string>>

描述：存储每个非终结符对应的 FOLLOW 集合。FOLLOW 集合包含可以出现在该非终结符右侧的终结符。

键是非终结符，值是该非终结符的 FOLLOW 集合。

2.6 VnIndex (非终结符索引)

类型：map<string, int>

描述：存储每个非终结符在分析表中的行索引。用于定位非终结符在分析表中的位置。

2.7 VtIndex (终结符索引)

类型：map<string, int>

描述：存储每个终结符在分析表中的列索引。用于定位终结符在分析表中的位置。

2.8 table (LL(1)分析表):

类型：vector<vector<string>>

描述：存储 LL(1) 分析表。每个非终结符与终结符的交集位置，存储对应的产生式。

如果某个位置为空字符串，表示该非终结符与该终结符不匹配（即该位置不允许该符号的推导）。

3. 程序结构描述

3.1 主程序 main()

功能：程序的主入口，初始化数据结构并执行文法分析过程。

输出：输出 Vn 集合、Vt 集合、FIRST 集合、FOLLOW 集合和 LL(1) 分析表。

结构描述：

从文法规则中构建 Vn 和 Vt 集合，记录非终结符和终结符的索引。

初始化并计算 FIRST 集合和 FOLLOW 集合。

构造 LL(1) 分析表。

读取并分析二元式文件，输出分析过程和结果。

3.2 getFirsts()

功能：计算并返回文法中每个非终结符的 FIRST 集合。

输出：返回 map<string, set<string>> 类型的 FIRST 集合，其中键是非终结符，值是该非终 结符的 FIRST 集合。

结构描述：

初始化每个非终结符的 FIRST 集合。

遍历文法产生式并根据产生式的右部推导 FIRST 集合：

如果产生式右部首个符号是终结符，直接加入到 FIRST 集合。

如果是非终结符，则递归添加该非终结符的 FIRST 集合。

不断更新直到 FIRST 集合不再变化。

3.3 getFollows()

功能：计算并返回文法中每个非终结符的 FOLLOW 集合。

输出：返回 map<string, set<string>> 类型的 FOLLOW 集合，其中键是非终结符，值是该 非终结符的 FOLLOW 集合。

结构描述：

初始化每个非终结符的 FOLLOW 集合，并将起始符号的 FOLLOW 集合设置为 #（结束符）。

遍历文法规则，递归地为每个非终结符计算 FOLLOW 集合：

如果一个非终结符后面跟随一个非终结符，则将该非终结符的 FIRST 集合添加到当前非终结符的 FOLLOW 集合中。

如果右部为空串，或右部的符号无法完全推导，则将该非终结符的 FOLLOW 集合传递给其左侧非终结符。

重复此过程，直到 FOLLOW 集合不再变化

3.4 getTable()

功能：构造 LL(1) 分析表。

输出：更新全局变量 table，即 LL(1) 分析表。

结构描述：

遍历文法产生式，对于每个产生式：

如果右部第一个符号是终结符，则将该产生式填入分析表相应位置。

如果右部第一个符号是非终结符，递归使用 FIRST 集合填充分析表。

如果右部为空串，则使用 FOLLOW 集合填充对应位置。

分析表通过 Vn 和 Vt 的索引确定每个位置的产生式。

3.5 analyze(const char expressionFile)*

功能：读取输入的二元式文件并进行 LL(1) 分析。

输出：打印 LL(1) 分析的过程，显示分析的结果（接受或拒绝）。

结构描述：

从文件中读取输入表达式，转换成符合文法要求的形式。

使用栈模拟 LL(1) 分析过程：

栈顶为非终结符时，从分析表中查找对应产生式并将右部压栈。

栈顶为终结符时，检查输入符号与栈顶符号是否匹配，若匹配则消费该符号并弹栈。

若栈和输入串都为空且分析过程未出现错误，则输出 "Accepted"，否则输出 "Rejected"。

4. 程序测试（包括选做内容）

4.1 LL(1)分析表和 first 集、follow 集构造结果

```
Vn集合：A E E' F M S T T' V

Vt集合：# ( ) * + - / = i

First集合：
A: "+" "-"
E: "(" "i"
E': "" "+" "-"
F: "(" "i"
M: "*" "/"
S: "i"
T: "(" "i"
T': "" "*" "/"
V: "i"

Follow集合：
A: "(" "i"
E: "#" ")"
E': "#" ")"
F: "#" ")" "*" "+" "-" "/"
M: "(" "i"
S: "#"
T: "#" ")" "+" "-"
T': "#" ")" "+" "-"
V: "="
```

```
*****************LL(1)分析表******************
        |#      |(      |)      |*      |+      |-      |/      |=      |i      |
A       |+      |-      |       |       |       |       |       |       |       |
E       |       |       |TE'    |       |TE'    |       |       |       |       |
E'      |ATE'   |ATE'   |       |       |       |       |       |       |       |
F       |       |       |(E)    |       |i      |       |       |       |       |
M       |       |       |       |       |       |*      |/      |       |       |
S       |       |       |       |       |V=E    |       |       |       |       |
T       |       |       |FT'    |       |FT'    |       |       |       |       |
T'      |       |       |       |       |       |MFT'   |MFT'   |       |       |
V       |       |       |       |       |i      |       |       |       |       |
```

4.2 简单表达式

a=b+c。程序会展现逐步的分析过程，包括使用的规则以及每一步的新产生式，完整过程如下。最终此例会输出 Accepted 表示识别成功。

```
输入语句为：
(2,a)
(4,=)
(2,b)
(4,+)
(2,c)
输入的语句为： a=b+c#
转化为： i=i+i#

LL(1)分析结果：

产生式： V=E
Push: E
Push: =
Push: V
V=E#                        i=i+i#

产生式： i
Push: i
i=E#                        i=i+i#
Match: i
=E#                         =i+i#
Match: =
E#                          i+i#

产生式： TE'
Push: '
Push: E
Push: T
```

```
产生式： FT'
Push: '
Push: T
Push: F
FT'E'#                          i+i#

产生式： i
Push: i
iT'E'#                          i+i#
Match: i
T'E'#                           +i#

产生式：
E'#                             +i#

产生式： ATE'
Push: '
Push: E
Push: T
Push: A
ATE'#                           +i#

产生式： +
Push: +
+TE'#                           +i#
Match: +
TE'#                            i#

产生式： FT'
```

```
产生式：FT'
Push: '
Push: T
Push: F
FT'E'#                    i#

产生式：i
Push: i
iT'E'#                    i#
Match: i
T'E'#                     #

产生式：
E'#                       #

产生式：
#                         #
Accepted
```

4.3 无效表达式

i=*i。是不符合文法的，因此输出 Rejected

```
输入语句为：
(2,a)
(4,=)
(4,*)
(2,i)
输入的语句为： a=*i#
转化为： i=*i#

LL(1)分析结果：

产生式：V=E
Push: E
Push: =
Push: V
V=E#                      i=*i#

产生式：i
Push: i
i=E#                      i=*i#
Match: i
=E#                       =*i#
Match: =
E#                        *i#

产生式：
#                         *i#
Rejected
```

4.4 带括号的表达式

a=(b+c)*d。符合文法规则，输出 Accepted。

4.5 空表达式

i=。符合文法规则，输出 Accepted。

4.6 较为复杂的综合运算表达式

```
输入语句为：
(2,i)
(4,=)
(4,()
(2,i)
(4,+)
(2,i)
(4,*)
(2,i)
(4,))
(4,-)
(2,i)
(4,*)
(4,()
(2,i)
(4,/)
(2,i)
(4,))
输入的语句为： i=(i+i*i)-i*(i/i)#
转化为： i=(i+i*i)-i*(i/i)#
```

```
产生式：
#                       #
Accepted
************************************************
```

5. 附录：完整代码

```
1. #define _CRT_SECURE_NO_WARNINGS 1
2. #include <iostream>
3. #include <fstream>
4. #include <set>
5. #include <map>
6. #include <vector>
7. #include <string>
8. #include<cstring>
9. using namespace std;
10.
11. // 定义文法规则
12. map<string, vector<string>> grammar = {
```

```
13.     {"S", {"V=E"}},
14.     {"E", {"TE\'"}},
15.     {"E\'", {"ATE\'", ""}},
16.     {"T", {"FT\'"}},
17.     {"T\'",{"MFT\'",""}},
18.     {"F",{"(E)","i"}},
19.     {"A", {"+", "-"}},
20.     {"M", {"*", "/"}},
21.     {"V", {"i"}}
22. };
23.
24. string line;
25. set<string> Vt; // 终结符集合
26. set<string> Vn; // 非终结符集合
27. map<string, set<string>> firsts; // FIRST 集合
28. map<string, set<string>> follows; // FOLLOW 集合
29. map<string, int>VnIndex; //分析表中 Vn 的下标
30. map<string, int>VtIndex; //分析表中 Vt 的下标
31. vector<vector<string>> table; // LL(1)分析表
32.
33.
34.
35. // 判断是否是非终结符
36. bool isVn(const string& symbol) {
37.     return Vn.count(symbol);
38. }
39.
40. // 判断是否是终结符
```

```
41. bool isVt(const string& symbol) {
42.     return Vt.count(symbol);
43. }
44. 
45. // 计算 First 集
46. map<string, set<string>> getFirsts() {
47.     map<string, set<string>> firstSet;
48. 
49.     // 初始化 First 集
50.     for (auto& entry : grammar) {
51.         firstSet[entry.first] = {};
52.         for (auto& production : entry.second) {
53.             string s = production;
54.             if (isVt(s.substr(0, 1))) {
55.                 firstSet[entry.first].insert(s.substr(0, 1));
56.             }
57.         }
58.     }
59. 
60.     bool changes = true;
61.     while (changes) {
62.         changes = false;
63. 
64.         // 遍历文法规则，计算 First 集
65.         for (auto& entry : grammar) {
66.             const string& nonTerminal = entry.first;
67.             int originalSize = firstSet[nonTerminal].size();
68. 
```

```
69.         for (auto& production : entry.second) {
70.             bool epsilonFlag = false;
71.             int i = 0;
72.
73.             // 遍历产生式右部的符号
74.             for (i = 0; i < production.length(); ++i) {
75.                 string symbol = production.substr(i, 1);
76.
77.                 if (isupper(production[i])) {  // 如果是非终结符
78.                     string vn = symbol;
79.                     if (production[i + 1] == '\'') {  // 如果是带 '\' 的非终结符
80.                         vn += "\'";
81.                         for (const auto& fi : firstSet[vn]) {
82.                             if (fi == "") epsilonFlag = true;
83.                             else firstSet[nonTerminal].insert(fi);
84.                         }
85.                         ++i;
86.                     }
87.                     else {
88.                         for (const auto& fi : firstSet[vn]) {
89.                             if (fi == "") epsilonFlag = true;
90.                             else firstSet[nonTerminal].insert(fi);
91.                         }
92.                     }
93.
94.                     // 如果没有 epsilon，就停止处理
95.                     if (!epsilonFlag) break;
96.                 }
97.                 else {  // 如果是终结符
```

```
                    firstSet[nonTerminal].insert(symbol);
                    break;
                }
            }

            // 如果产生式右部可以推导出 ε，加入 ε
            if (i == production.length()) {
                firstSet[nonTerminal].insert("");
            }
        }

        // 如果 First 集更新了，继续迭代
        if (firstSet[nonTerminal].size() != originalSize) {
            changes = true;
        }
    }
}

return firstSet;
}

// 计算 Follow 集
map<string, set<string>> getFollows()
{
    map<string, set<string>> followSet;
    map<string, int> originalSize; //前一次迭代 follow 集大小

```

```
126.    // 初始化非终结符的 Follow 集
127.    for (auto const& entry : grammar)
128.    {
129.        followSet[entry.first] = {};
130.    }
131.    followSet["S"].insert("#");
132.
133.    bool changes = true;
134.    while (changes)
135.    {
136.        changes = false;
137.        for (map<string, vector<string>>::iterator entry = grammar.begin(); entry != gram
        mar.end(); entry++)
138.        {
139.            string nonTerminal = entry->first;
140.            originalSize[nonTerminal] = followSet[nonTerminal].size();
141.            for (vector<string>::iterator it = entry->second.begin(); it != entry->second.end();
        it++)
142.            {
143.                string production = *it;
144.                for (int i = 0; i < production.size(); i++)
145.                {
146.                    if (isupper(production[i]))// 如果是非终结符
147.                    {
148.                        string vn = " ";
149.                        vn[0] = production[i];
150.
151.                        if (production[i + 1] == '\'')
152.                        {
```

```
153.                 vn = vn + "\'";
154.                 i++;
155.             }
156.
157.             string las = production.substr(i + 1);
158.             if (las != "")
159.             {
160.                 int j;
161.                 for (j = 0; j < las.length(); j++)
162.                 {
163.                     if (isupper(las[j])) // 如果是非终结符
164.                     {
165.                         string vnn = " ";
166.                         bool flag = false;
167.                         vnn[0] = las[j];
168.
169.                         if (las[j + 1] == '\'')
170.                         {
171.                             vnn = vnn + "\'";
172.                             j++;
173.                         }
174.
175.                         for (set<string>::iterator fi = firsts[vnn].begin(); fi != firsts[vnn].e
    nd(); fi++)
176.                         {
177.                             if (*fi == "")
178.                                 flag = true;
179.                             else
180.                                 followSet[vn].insert(*fi);
```

```
181.                         }
182.
183.                         if (!flag)
184.                             break;
185.
186.                     }
187.                     else
188.                     {
189.                         followSet[vn].insert(string(1, las[j]));
190.                         break;
191.                     }
192.                 }
193.
194.                 if (j == las.length())
195.                 {
196.                     for (set<string>::iterator fo = followSet[nonTerminal].begin(); fo !=
    followSet[nonTerminal].end(); fo++)
197.                         followSet[vn].insert(*fo);
198.                 }
199.             }
200.             else
201.             {
202.                 for (set<string>::iterator fo = followSet[nonTerminal].begin(); fo != fol
    lowSet[nonTerminal].end(); fo++)
203.                     followSet[vn].insert(*fo);
204.             }
205.         }
206.     }
207. }
```

```
208.        }
209.        for (map<string, vector<string>>::iterator entry = grammar.begin(); entry != gram
    mar.end(); entry++)
210.        {
211.            if (followSet[entry->first].size() != originalSize[entry->first]) //若follow 改变，
    则继续迭代
212.            {
213.                changes = true;
214.                break;
215.            }
216.        }
217.    }
218.
219.    return followSet;
220. }
221.
222. // 构造 LL(1)分析表
223. void getTable()
224. {
225.    for (map<string, vector<string>>::iterator it = grammar.begin(); it != grammar.end(); i
    t++)
226.    {
227.        int vnidx = VnIndex[it->first];
228.        for (vector<string>::iterator tmp = it->second.begin(); tmp != it->second.end(); tmp
    ++)
229.        {
230.            string str = *tmp; // 右部
231.            if (str == "")    //为空则将follow(left)中元素填入 left->^
232.            {
```

```
233.             for (set<string>::iterator b = follows[it->first].begin(); b != follows[it->first].e
    nd(); b++)
234.             {
235.                 if (isVt(*b))
236.                 {
237.                     table[vnidx][VtIndex[*b]] = "";
238.                 }
239.             }
240.         }
241.         else if (isVt(str.substr(0, 1))) //产生式右部第一个字符是终结符
242.             table[vnidx][VtIndex[str.substr(0, 1)]] = str;
243.         else //产生式右部第一个字符是非终结符，遍历其first 集
244.         {
245.             string::iterator iit;
246.             for (iit = str.begin(); iit != str.end();)
247.             {
248.                 string vn = " ";
249.                 bool flag = false;
250.                 vn[0] = *iit;
251.                 iit++;
252.                 if (*iit == '\'')
253.                 {
254.                     vn = vn + "\'";
255.                     iit++;
256.                 }
257.
258.                 for (set<string>::iterator fi = firsts[vn].begin(); fi != firsts[vn].end(); fi++)
259.                 {
260.                     if (*fi == "")
```

```
261.                     flag = true;
262.                 else
263.                     table[vnidx][VtIndex[*fi]] = str;
264.             }
265.             if (!flag)
266.                 break;
267.         }
268.
269.         if (iit == str.end())//遍历其follow 集合
270.         {
271.             for (set<string>::iterator b = follows[it->first].begin(); b != follows[it->first].
    end(); b++)
272.             {
273.                 if (isVt(*b))
274.                 {
275.                     table[vnidx][VtIndex[*b]] = "";
276.                 }
277.             }
278.         }
279.
280.     }
281.   }
282. }
283. }
284.
285. // 读取二元式文件并进行 LL(1)分析
286. void analyze(const char* expressionFile)
287. {
288.     printf("\n");
```

```
289.    cout << "输入语句为：" << endl;
290.    FILE* fp;
291.    char buf[1024];
292.    string shizi, like;
293.    if ((fp = fopen(expressionFile, "r")) != NULL)
294.    {
295.        while (fgets(buf, 1024, fp) != NULL)
296.        {
297.            int len = strlen(buf);
298.            buf[len - 1] = '\0';  /*去掉换行符*/
299.            printf("%s \n", buf);
300.
301.            if (buf[1] == '2') // 说明为标识符
302.            {
303.                like += 'i';
304.            }
305.            for (int i = 3; i < len - 2; i++)
306.            {
307.                shizi = shizi + buf[i];
308.                if (buf[1] != '2')
309.                {
310.                    like += buf[i];
311.                }
312.            }
313.        }
314.    }
315.    shizi += '#';
316.    like += '#';
317.    fclose(fp);
```

```
318.
319.    cout << "输入的语句为：" << shizi << endl;
320.    cout << "转化为：" << like << endl;
321.    printf("\n");
322.
323.    cout << "LL(1)分析结果：" << endl;
324.
325.    string expression = like;
326.    string result = "ACCEPT";
327.    string topOfStack = "S#";
328.
329.    while (topOfStack.size() > 1 && result == "ACCEPT")
330.    {
331.        char nextSymbol = expression[0];
332.
333.        if (isVt(topOfStack.substr(0, 1)))
334.        {
335.            if (nextSymbol == topOfStack[0])
336.            {
337.                cout << "Match: " << nextSymbol << endl;
338.                expression = expression.substr(1);
339.                topOfStack = topOfStack.substr(1);
340.            }
341.            else
342.            {
343.                result = "REJECT";
344.                cout << "Error: Mismatch for symbol " << nextSymbol << endl;
345.            }
```

```
346.        }
347.        else
348.        {
349.            string vn = " ", vt = " ";
350.            int idx = 0;
351.
352.            vn[0] = topOfStack[idx++];
353.            vt[0] = nextSymbol;
354.            if (topOfStack[idx] == '\')
355.                vn = vn + topOfStack[idx++];
356.
357.            string production = table[VnIndex[vn]][VtIndex[vt]];
358.            printf("\n");
359.            cout << "产生式: " << production << endl;
360.
361.            if (production != "")
362.            {
363.                // 非空产生式，将产生式右部压入栈
364.                topOfStack = topOfStack.substr(idx);
365.                for (int i = production.length() - 1; i >= 0; i--)
366.                {
367.                    cout << "Push: " << production[i] << endl;
368.                    topOfStack = production[i] + topOfStack;
369.                }
370.            }
371.            else
372.                topOfStack = topOfStack.substr(idx);
373.        }
374.        cout << topOfStack << "\t\t\t" << expression << endl;
```

```
375.     }
376.
377.     if (topOfStack.size() == 1 && expression.size() == 1 && result == "ACCEPT")
378.     {
379.         cout << "Accepted" << endl;
380.     }
381.     else
382.     {
383.         cout << "Rejected" << endl;
384.     }
385.
386.     cout << "*********************************************" << endl;
387.     printf("\n");
388. }
389.
390. int main()
391. {
392.     int iVt = 0, iVn = 0;
393.
394.     // 记录 Vn 集合和 Vt 集合及其在分析表中的下标
395.     for (map<string, vector<string>>::iterator it = grammar.begin(); it != grammar.end();
        ++it)
396.     {
397.         pair<string, vector<string>> temp = *it;
398.
399.         Vn.insert(temp.first);
400.         VnIndex[temp.first] = iVn++;
401.
```

```
402.        vector<string> production = temp.second;
403.        for (vector<string>::iterator iit = production.begin(); iit != production.end(); iit++)
404.        {
405.            string ss = *iit;
406.            for (int j = 0; j < ss.length(); j++)
407.            {
408.                //记录产生式中的非终结符和终结符
409.                if (ss[j] >= 'A' && ss[j] <= 'Z')
410.                { // 大写字母
411.                    if (ss[j + 1] == '\'')
412.                    {       // 有'则读入俩作为一个非终结符
413.                        Vn.insert(ss.substr(j, 2));
414.                        j++;
415.                    }
416.                    else
417.                    {
418.                        Vn.insert(ss.substr(j, 1));
419.                    }
420.                }
421.                else
422.                {
423.                    Vt.insert(ss.substr(j, 1)); // 是终结符
424.                    string v = ss.substr(j, 1);
425.                    if (v != "" && VtIndex[v] == 0)
426.                        VtIndex[v] = iVt++;
427.                }
428.            }
429.
430.        }
```

```
431.    }
432.    Vt.insert("#");
433.    VtIndex["#"] = iVt++;
434.    table.resize(Vn.size(), vector<string>(Vt.size(), ""));
435.
436.    /***********求解部分***********/
437.
438.    // 求first 集合
439.    firsts = getFirsts();
440.    // 求follow 集合
441.    follows = getFollows();
442.    // 构造 LL(1)分析表
443.    getTable();
444.    // 读取二元式文件并进行 LL(1)分析
445.    analyze("D:/lexical_analysis_output.txt");
446.
447.
448.    /***********输出部分***********/
449.
450.    // 输出 Vn 集合
451.    cout << "Vn 集合: ";
452.    for (set<string>::iterator it = Vn.begin(); it != Vn.end(); it++)
453.        cout << *it << ' ';
454.    cout << endl;
455.    printf("\n");
456.
457.    // 输出 Vt 集合
458.    cout << "Vt 集合: ";
```

```
459.    for (set<string>::iterator it = Vt.begin(); it != Vt.end(); it++)
460.        cout << *it << ' ';
461.    cout << endl;
462.    printf("\n");
463.
464.
465.    // 输出 First 集合
466.    cout << "First 集合: " << endl;
467.    for (map<string, set<string>>::iterator it = firsts.begin(); it != firsts.end(); it++)
468.    {
469.        cout << it->first << ": ";
470.        for (set<string>::iterator it2 = it->second.begin(); it2 != it->second.end(); it2++) {
471.            cout << '\'' << *it2 << '\'' << " ";
472.        }
473.        cout << endl;
474.    }
475.    printf("\n");
476.
477.
478.    // 输出 Follow 集合
479.    cout << "Follow 集合: " << endl;
480.    for (map<string, set<string>>::iterator it = follows.begin(); it != follows.end(); it++) {
481.        cout << it->first << ": ";
482.        for (set<string>::iterator it2 = it->second.begin(); it2 != it->second.end(); it2++) {
483.            cout << '\'' << *it2 << '\'' << " ";
484.        }
485.        cout << endl;
486.    }
```

```
487.    printf("\n");
488.
489.    // 输出 LL(1) 分析表
490.    cout << "*****************LL(1)分析表******************" << endl;
491.    //cout << "----------------------------------------" << endl;
492.    cout << "\t|";
493.    for (set<string>::iterator vt = Vt.begin(); vt != Vt.end(); vt++)
494.    {
495.        cout << *vt << "\t|";
496.    }
497.    cout << endl;
498.    for (map<string, vector<string>>::iterator it = grammar.begin(); it != grammar.end(); i
        t++)
499.    {
500.        cout << it->first << "\t|";
501.        for (int j = 0; j < Vt.size(); j++)
502.        {
503.            cout << table[VnIndex[it->first]][j] << "\t|";
504.        }
505.        cout << endl;
506.    }
507.
508.    return 0;
509.
510. }
```