编号：

**《编译原理》实验报告**

**题 目 ： 自上而下语法分析**

**姓 名 ：**

**学 号 ：**

**院 （系）： 计算机科学与技术**

**指导教师：**

**成 绩 ：**

**2024年 11月 28日**

1. **实验目的**

（1）理解和掌握把问题中的实体转换成抽象模型中数据结构的能力；分析文法概念，建立文法描述的数据结构，实现 First 集、Follow 集和 Select 集的计算，加深对文法的理解。

（2）理解LL语法分析方法的原理，掌握LL语法分析器的构造，设计分析器数据结构和程序结构，加深对自上而下语法分析方法的理解。

1. **问题描述**

我实现了一个实现一个 LL(1)语法分析器。LL(1)分析器是一种自上而下的语法分析方法，它通过预测和递归下降来分析给定的输入符号串是否符合指定的上下文无关文法（CFG）。

**本实验需要实现的功能有：**

1. 构造文法的 LL 预测分析表；

2. 构造 LL 语法分析器的总控程序；

3. 输入文法：文法描述存储在文本文件中（编码格式 ANSI），文件名作为命令行参数输入；

4. 输入待分析的符号串：符号串存储在文本文件中（编码格式 ANSI），文件名作为命令行参数输入。

5. 输出文法的 LL 预测分析表到标准输出设备；

6. 输出分析结果：输出待分析符号串是否为该文法正确句子的判断，并输出文本形式的分析过程（标准输出设备）。

**实现原理和方法：**

1. 构建first集：对于任意一个非终结符A，如果A能推导出一个以终结符开始的符号串，则该终结符加入A的FIRST集；推导一个非终结符的符号，则带入生成式；推导出ε，则加入下一个符号的first集，
2. 构建 FOLLOW 集：对于每个产生式A->αBβ，如果B是非终结符，且β能推导出空串，则将FOLLOW(A)中的终结符加入FOLLOW(B)。若B在产生式的右部为最后一个符号，则将 FOLLOW(A)中的终结符加入FOLLOW(B)。反之，将下一个符号的first集加入FOLLOW(B)中。

3. 构建SELECT集：给定上下文无关文法的产生式A→α, A∈VN,α∈V, 若α不能推导出ε,则SELECT(A→α)=FIRST(α)。如果α能推导出ε则：SELECT(A→α)=(FIRST(α) –{ε})∪FOLLOW(A)

4. 构建预测分析表(Parsing Table)：使用SELECT集来填充分析表。如果产生式A->α的SELECT(A>-α)包含终结符t，则在预测分析表的(A, t)位置放置产生式A->α。

5. 符号串分析过程：利用栈模拟自上而下的分析过程。栈用于存储待分析的符号，输入符号串从队列中逐个取出。

(1)如果栈顶符号和当前输入符号相同，则出栈并继续分析。如果栈顶符号是非终结符，则查表使用相应的产生式进行推导，逆序将产生式右部符号压入栈中。

(2)如果栈顶符号与输入符号不匹配，或者无法在分析表中找到对应的产生式，分析失败。

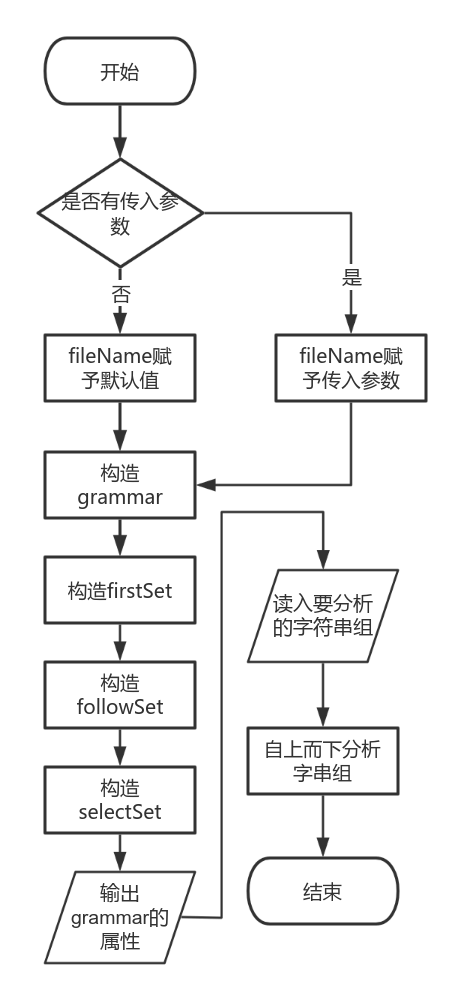
1. **软件设计方法的选择**

**表1 软件设计方法及开发语言和环境**

|  |  |
| --- | --- |
| **软件设计方法** | **结构化设计方法** |
| 开发语言 | C++ |
| 开发环境 | windows11，Visual Studio 2022 |

**表2 创建的模型**

|  |  |
| --- | --- |
| **分析模型** | **设计模型** |
| 系统流程图、数据字典 | 类图、函数接口 |

1. **分析模型**

**系统流程图：**

整个系统的流程可以分为几个步骤，首先，通过readGrammar函数读取并解析输入的文法文件，提取出文法中的非终结符、终结符、产生式以及起始符号等信息。这些信息将用于后续的语法分析和预测分析表的构建。在获取了这些基础信息后，系统会通过getFirst、getFollow函数计算每个非终结符的First和Follow集。基于First集和Follow集的信息，系统会进一步在getSelect函数中计算每个产生式的Select集。

在计算出所有必要的集合后，系统会通过getTable函数生成LL(1)预测分析表。这个表是语法分析的核心，包含了根据当前栈顶符号和输入符号，应该选择的产生式。当所有准备工作完成后，系统通过analyseStr函数开始进行实际的语法分析。在分析过程中，输入符号串会被加载到队列中，起始符号会被压入栈中。系统会逐步从栈中弹出符号，并与队列中的当前输入符号进行匹配。如果栈顶符号和输入符号相同，表示匹配成功，输入指针后移；如果不相同，则通过预测分析表查找对应的产生式，将产生式的右部逆序压入栈中，继续进行分析。如果某个步骤匹配失败，系统会输出错误信息并终止分析。整个分析过程中，系统会详细输出每一步的操作步骤，使用户能够清楚地了解语法分析的每个细节。最终，如果输入符号串符合文法，系统输出匹配成功的信息，否则会输出匹配失败的错误信息。

这个系统的核心在于通过LL(1)文法分析算法，结 **图1 自上而下语法分析器**

合First集、Follow集、Select集和预测分析表来有效地实现对输入符号串的语法分析，确保文法的正确性和语法分析的精确性。

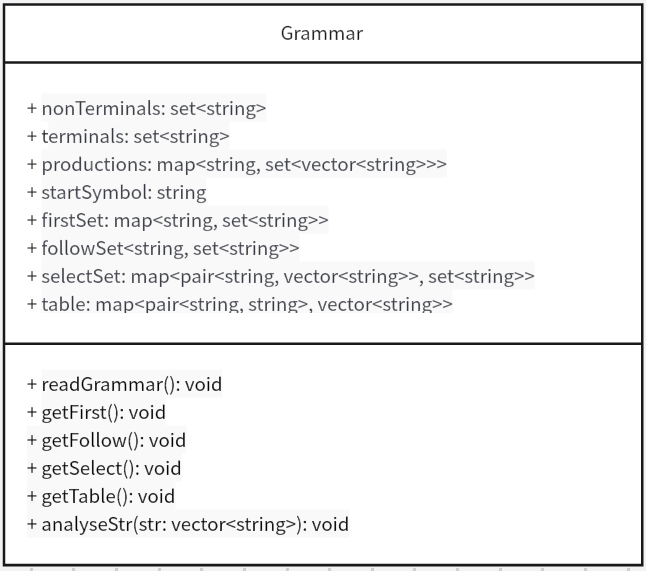
**数据字典：**

**表3 数据字典**

|  |  |  |
| --- | --- | --- |
| **数据名称** | 数据类型 | 数据作用 |
| **nonTerminals** | set<string> | 文法中的所有非终结符 |
| **terminals** | set<string> | 文法中的所有终结符 |
| **productions** | map<string, set<vector<string>>> | 映射每个非终结符到其对应的产生式集合 |
| **startSymbol** | string | 文法的起始符号 |
| **firstSet** | map<string, set<string>> | 每个非终结符的First集 |
| **followSet** | map<string, set<string>> | 每个非终结符的Follow集 |
| **selectSet** | map<pair<string,vector<string>>,set<string>> | 每个产生式的Select集 |
| **table** | map<pair<string, string>, vector<string>> | LL(1) 预测分析表 |

1. **设计模型**

**类图：**

**Grammar类：**

Grammar类是实现 LL(1) 语法分析器的核心，它负责管理文法信息并执行相关的计算与分析操作。类内部包含多个成员变量，包括非终结符集合、终结符集合、产生式集合、起始符号、以及用于语法分析的 FIRST 集、FOLLOW 集、SELECT 集等数据结构。通过构造函数，Grammar 类能够从外部文件读取文法定义，并将文法的各个组成部分存储到相应的成员变量中。类的主要功能包括计算 FIRST 集、FOLLOW 集和 SELECT 集，构建预测分析表，并通过分析输入的符号串来判断它是否符合文法。通过 getFirst、getFollow、getSelect 等方法，Grammar 类能够递归地计算文法中每个非终结符的 FIRST 和 FOLLOW 集， **图2 Grammar类**

进而生成 SELECT 集并填充预测分析表。分析过程通过栈和队列的配合实现，模拟 LL(1) 分析的逐步推导。analyseStr 方法不仅会输出分析过程的每一步，还能在遇到无法匹配的情况时给出错误反馈。总的来说，Grammar 类通过面向对象的设计思想将文法的表示、分析过程和预测表的生成高度封装，使得 LL(1) 分析过程更加清晰、易于维护与扩展。

**函数接口：**

**表4 函数接口**

|  |  |  |  |
| --- | --- | --- | --- |
| **函数名** | 传入参数类型 | 返回参数类型 | 描述 |
| **readGrammar** | 无 | void | 读取并解析文法文件 |
| **getFirst** | 无 | void | 计算各非终结符First集 |
| **getFollow** | 无 | void | 计算各非终结符Follow集 |
| **getSelect** | 无 | void | 计算各非终结符Select集 |
| **getTable** | 无 | void | 生成LL(1)预测分析表 |
| **analyseStr** | vector<string> | void | 进行语法分析，逐步对输入符号串进行匹配、产生式推导 |

1. **主要算法描述**

**（1）求first集（如下图3）**

Grammar::getFirst()方法的目的是计算文法中每个非终结符的 FIRST 集。首先，定义一个布尔变量flag用于控制迭代过程，并初始化一个临时的firstTempSet来存储后续递归更新关系。在最初的计算中，遍历每个非终结符及其对应的产生式，对于产生式的右部中的每个符号，如果符号是终结符或空串ε，则直接将该符号加入对应非终结符的FIRST集。如果右部的符号是非终结符，则需要检查该非终结符的产生式是否包含空串ε，如果包含ε，则标记为需要进一步计算。接下来，对于每个非终结符，继续检查其右部其他符号的FIRST集，直到遇到一个不能推导空串的符号或者遍历完所有符号为止。在这个过程中，使用一个标志变量veri来判断是否需要继续递归地添加空串相关的符号。当第一次计算完所有非终结符的FIRST 集后，进入一个迭代过程，直到没有新的元素加入FIRST集为止。在每次迭代中，检查firstTempSet中的结果，将新计算的 FIRST 集合并到原来的集合中，直到没有更多的变化。最终，所有非终结符的 FIRST 集都得到了正确计算，并保存在firtSet中。

**（2）求follow集（如下图5）**

Grammar::getFollow()方法的目的是计算文法中每个非终结符的FOLLOW集。首先，方法初始化了一个空的followTempSet用于存储存储后续递归更新关系。接着将起始符号的 FOLLOW 集加入终结符 ，因为在 LL(1) 语法分析中，起始符号的 FOLLOW 集总是包含输入的结束标志符号 。接下来，遍历所有非终结符 left 和它们的产生式 vect，对每个产生式右部中的符号进行分析。如果遇到一个非终结符符号vect[i]，则检查它后面是否有符号（即 vect[i+1]）来决定如何更新 FOLLOW 集。如果vect[i]后面没有符号（即该符号是产生式的最后一个符号），则将产生该产生式的左部符号 left 加入到vect[i]的FOLLOW集中，这代表vect[i]后面可以跟随非终结符left。如果vect[i]后面是终结符，则将该终结符加入到vect[i]的 FOLLOW 集中。如果vect[i]后面是非终结符符号，且该符号的 FIRST 集不包含空串ε，则将 FIRST 集中的符号加入到 vect[i] 的FOLLOW集。如果该符号的FIRST集包含ε，则需要继续递归地分析后续符号，并将FOLLOW集中的符号加入到vect[i]的FOLLOW集。

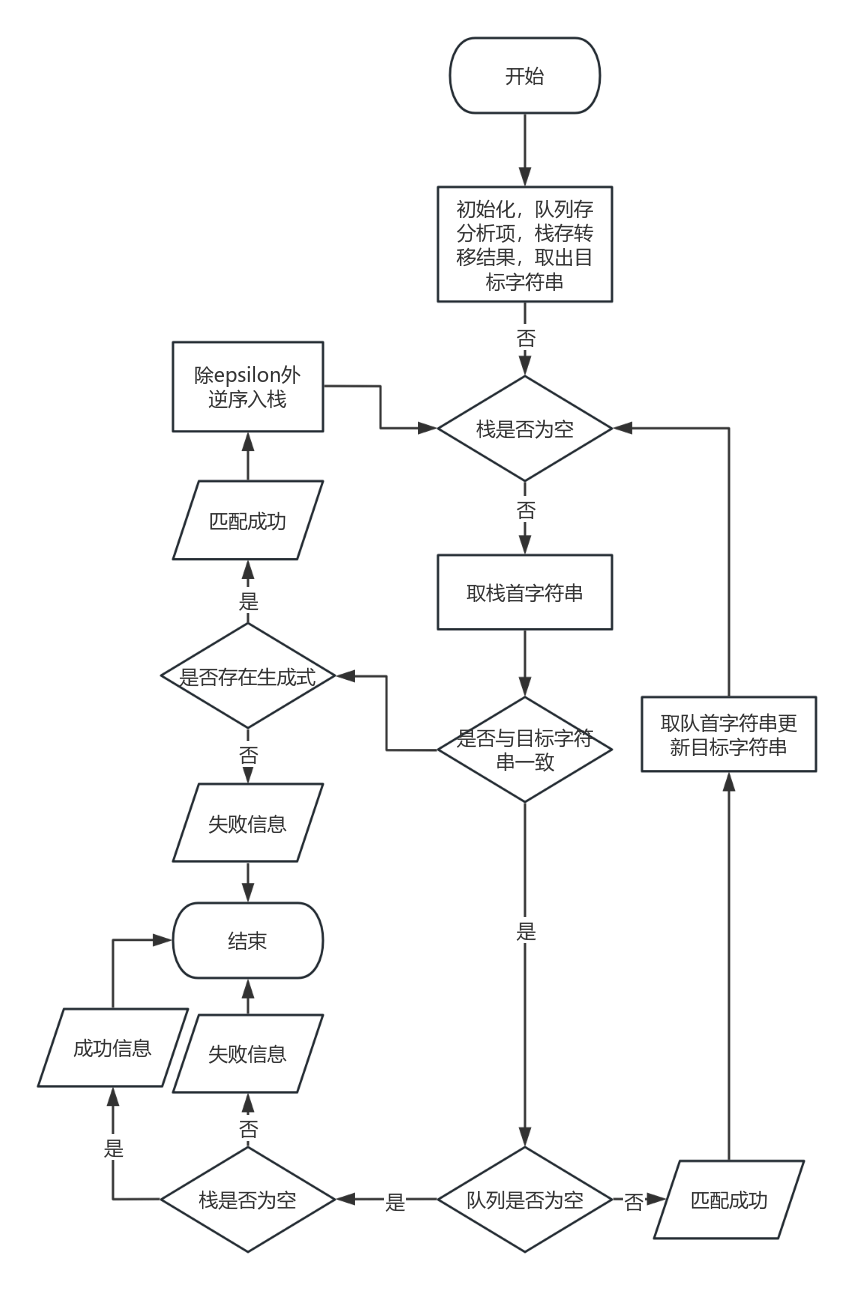
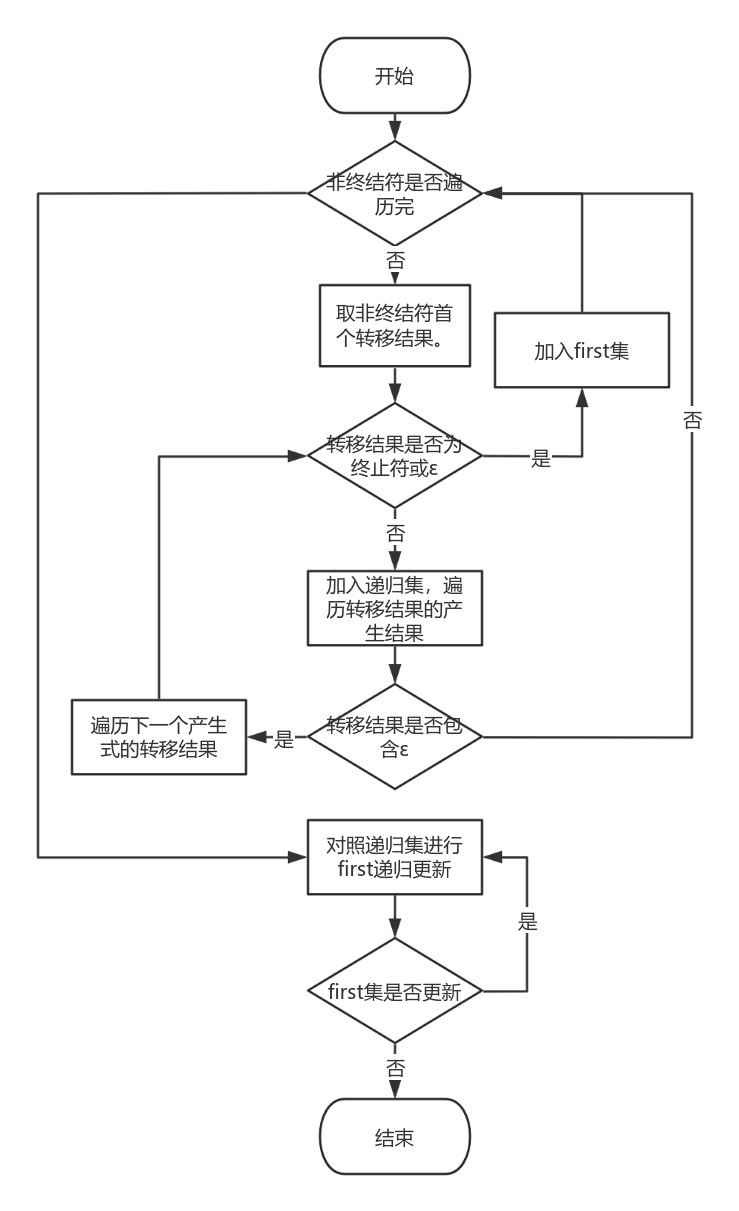
接下来，方法进入一个迭代过程，直到 FOLLOW 集不再发生变化。在每次迭代中，对于每个非终结符 left，检查其FOLLOW集是否有更新。如果有更新，就继续迭代计算。最终，当没有新的元素被添加到任何FOLLOW集时，算法结束，得到了每个非终结符的完整FOLLOW集。

**（3）求select集（如下图6）**

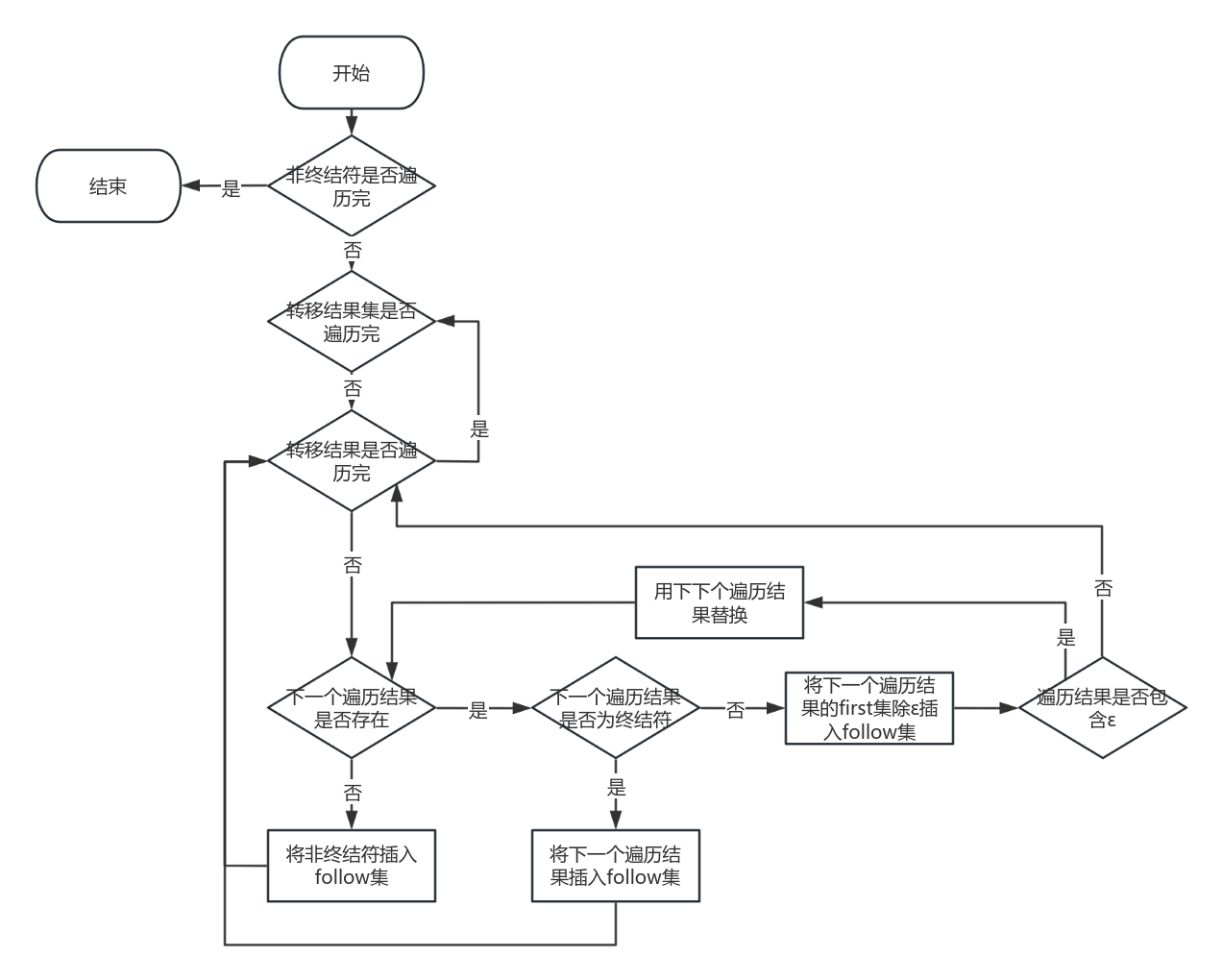
getSelect()方法计算每个产生式的SELECT集，主要用于LL(1)分析中的预测分析表构造。对于每个产生式A→X1 X2...Xn，首先检查右部的第一个符号X1。如果X1是终结符，直接将其加入SELECT集；如果X1是非终结符，则将其FIRST集中的符号加入SELECT集。如果X1可以推导出空串ε，则继续检查后续符号，直到找到一个非ε的终结符，或将FOLLOW集中的符号加入SELECT 集。如果产生式右部是ε，则将FOLLOW集中的符号加入SELECT集。最终，SELECT集为每个产生式提供了在预测分析表中使用的信息，帮助确定分析过程中应该使用哪个产生式。

**（4）求分析流程（如下图4）**

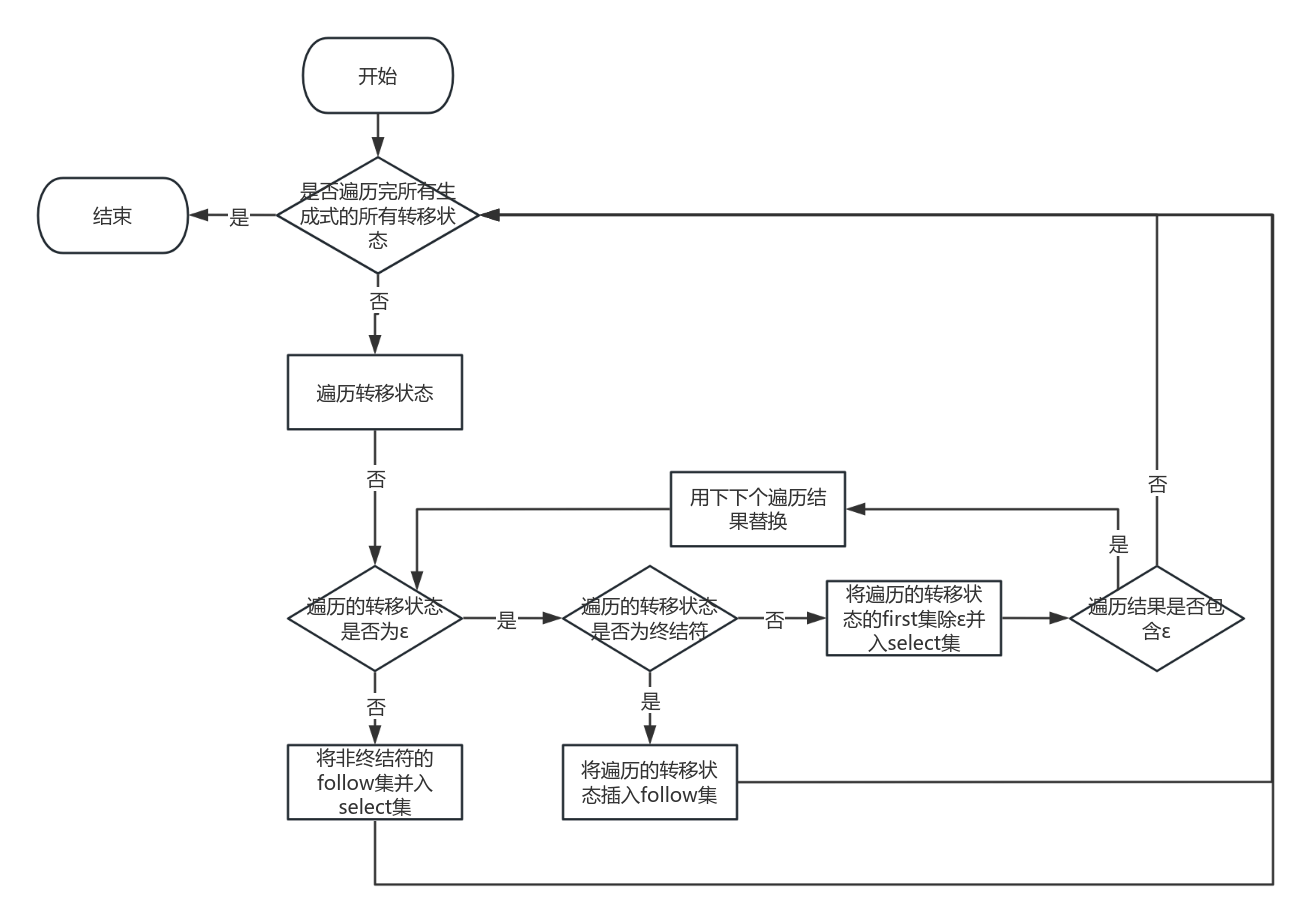
analyseStr()方法实现了基于LL(1)分析算法的语法分析过程。首先，将待分析的符号串按顺序存入队列，并将起始符号入栈。接下来，进入循环，直到栈为空。每次从栈顶取出一个符号进行处理。如果栈顶符号与队列头部的输入符号匹配，则将输入指针后移，继续分析。如果不匹配，则根据LL(1)预测分析表查找当前栈顶符号和输入符号对应的产生式，并将产生式的右部逆序入栈。如果当前栈顶符号与输入符号不匹配且无法找到对应的产生式，则分析失败，终止。整个过程会输出每一步的详细信息，包括栈的操作、输入符号的匹配情况、预测分析表的查找结果等，以帮助跟踪分析过程。如果成功匹配并且栈空，说明符号串符合文法，否则输出匹配失败的信息。



**图3 first集生成** **图4 分析流程**

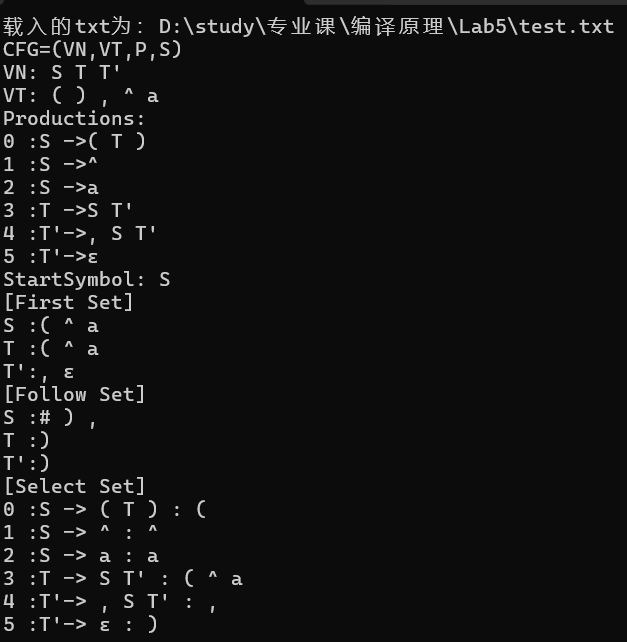
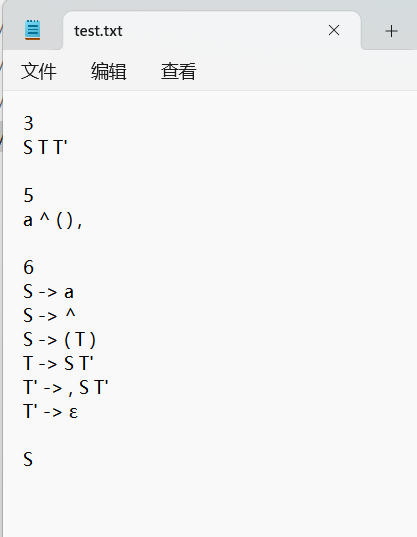


**图5 求follow集**

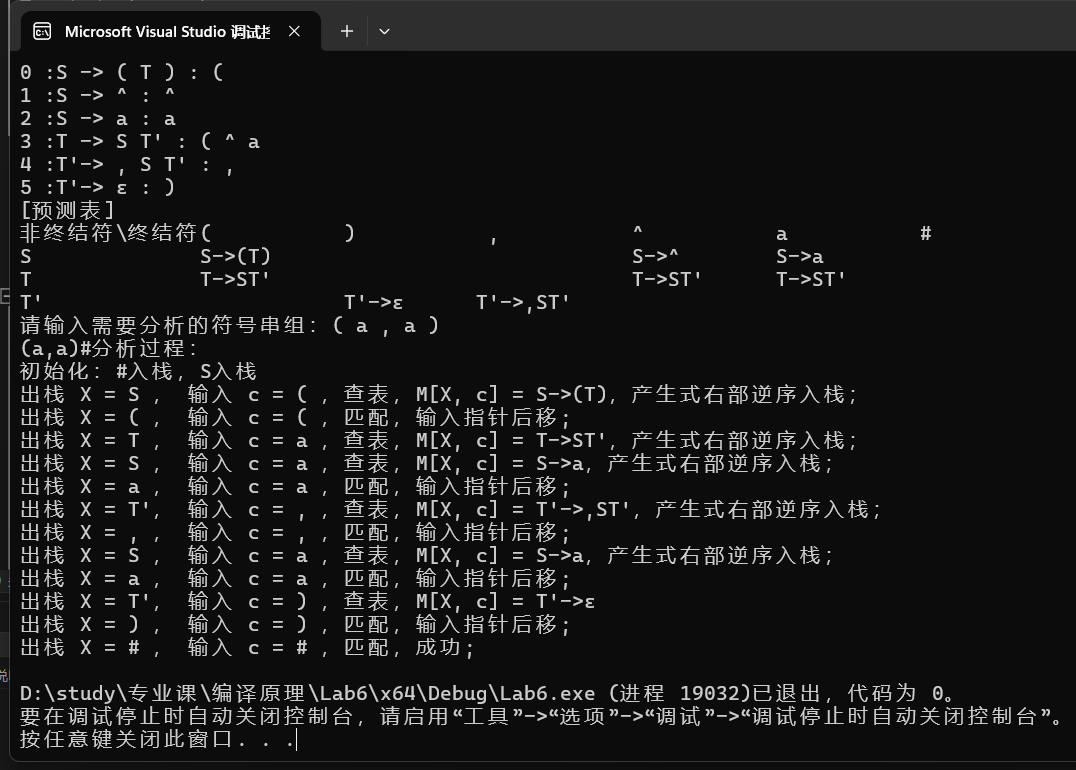


**图6 求select集**

1. **测试数据与测试效果**
2. **first、follow、select集生成：**



1. **词法分析：**



1. **实验总结**

在本次实验中，最大的挑战在于如何准确处理文法中的特殊情况，例如递归定义的产生式、包含ε的规则以及多个非终结符之间的复杂依赖关系。在计算First和Follow集时，尤其需要关注递归调用的终止条件和集合更新的准确性。在调试过程中，我遇到过 First 集无法正确更新的问题，最终通过在循环中加入集合大小的动态判断解决了此问题。这使我意识到，算法设计不仅要考虑逻辑正确性，还要保证其在边界情况下的稳定性。

在生成LL(1)预测分析表时，我发现Select集的构造对表的正确性至关重要，但起初忽略了 Follow 集对 Select 集构造的补充作用。通过详细分析文法规则和对照理论，我调整了 Select 集的生成逻辑，最终成功构造出正确的分析表。这一过程让我更加理解集合间的相互关系，也提高了理论与实践结合的能力。

在语法分析的实现过程中，匹配失败的错误处理是一个难点。在栈和队列的动态操作中，我多次遇到匹配失败却未能及时终止的问题，最终通过在分析表查找失败时加入错误提示逻辑，并详细输出每一步操作步骤，大幅提高了调试效率。这一过程让我认识到，程序的鲁棒性与用户可读性同样重要。

通过此次实验，我不仅掌握了 LL(1) 文法分析的理论与实现，也深刻体会到良好设计和细致调试的重要性。建议在今后的实验中，可以通过分步骤测试各个模块的正确性，逐步排查问题，并且注重记录每一步的思路和调整过程，以提升解决问题的效率。这次实验为后续编译器的学习打下了坚实的基础，也培养了我解决复杂问题的耐心和逻辑思维能力。