编号：

**《编译原理》实验报告**

**题 目 ： 词法分析**

**姓 名 ：**

**学 号 ：**

**院 （系）： 计算机科学与技术**

**指导教师：**

**成 绩 ：**

**2024年 11月 24日**

1. **实验目的**

通过本实验，学习和掌握词法分析器的基本构成和实现原理。具体目标包括：

1.设计与实现一个词法分析器，加深对词法分析原理的理解。

2.理解和掌握正则表达式RE到非确定有穷自动机NFA再到确定有穷自动机DFA的转换方法，加深对正则表达式和自动机的理解。

3.熟悉Lex工具自动生成词法分析器的使用。

实验完成后，我实现了一个完整的词法分析器，能够识别给定的源代码中的各种词法单元，并能够处理正则表达式，将其转换为NFA，进一步转换为DFA，并进行最小化处理。

1. **问题描述**

词法分析器是编译器的前端部分，负责将源代码文本分解成一系列的词法单元 。

**本实验需要实现的功能有：**

1.提取并分类词法单元。

2.将RE转为NFA，再将NFA转换成DFA并最小化。

3.基于 Lex 的词法分析器设计：设计并实现一个基于 Lex 的词法分析器。

**实现原理和方法：**

1.基于正则表达式和有限状态机的概念来实现词法单元的划分和识别。分割的词法单元分为关键字、专用符号、数字、乱码和变量名。其中数字、乱码和变量名的种别码分别为41、42、43，关键词和专用符号的种别码如表1、表2。

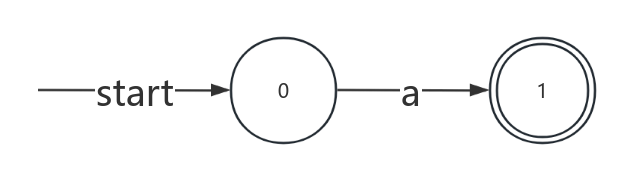
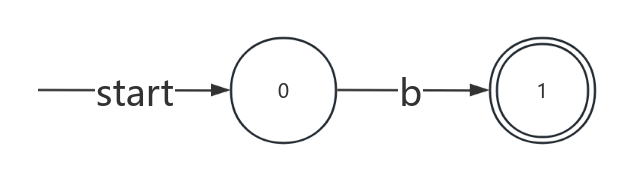
**表1 关键字**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 符号 | 种别码 | 符号 | 种别码 | 符号 | 种别码 | 符号 | 种别码 |
| main | 1 | to | 11 | bool | 21 | struct | 31 |
| if | 2 | step | 12 | and | 22 | break | 32 |
| then | 3 | switch | 13 | or | 23 | long | 33 |
| else | 4 | of | 14 | not | 24 | typedef | 34 |
| while | 5 | case | 15 | mod | 25 | const | 35 |
| do | 6 | default | 16 | read | 26 | float | 36 |
| repeat | 7 | return | 17 | write | 27 | short | 37 |
| until | 8 | integer | 18 | static | 28 | continue | 38 |
| for | 9 | real | 19 | int | 29 | void | 39 |
| from | 10 | char | 20 | double | 30 | sizeof | 40 |

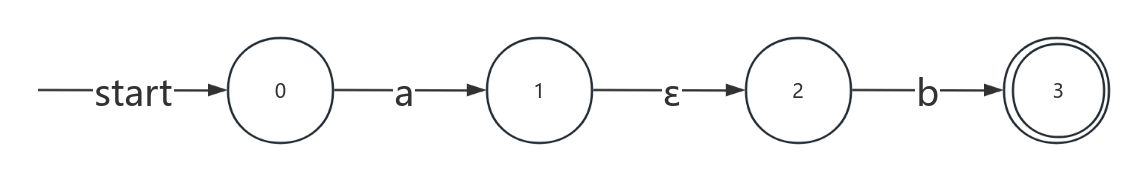
**表2 专用符号**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 符号 | 种别码 | 符号 | 种别码 | 符号 | 种别码 | 符号 | 种别码 |
| - | 44 | <= | 51 | { | 60 | += | 67 |
| \* | 45 | > | 52 | } | 61 | \*= | 68 |
| / | 46 | >= | 53 | [ | 62 | /= | 69 |
| : | 47 | = | 54 | ] | 63 | -= | 70 |
| := | 48 | ; | 57 | ++ | 64 | ' | 71 |
| < | 49 | ( | 58 | , | 65 | == | 72 |
| != | 50 | ) | 59 | -- | 66 | + | 73 |

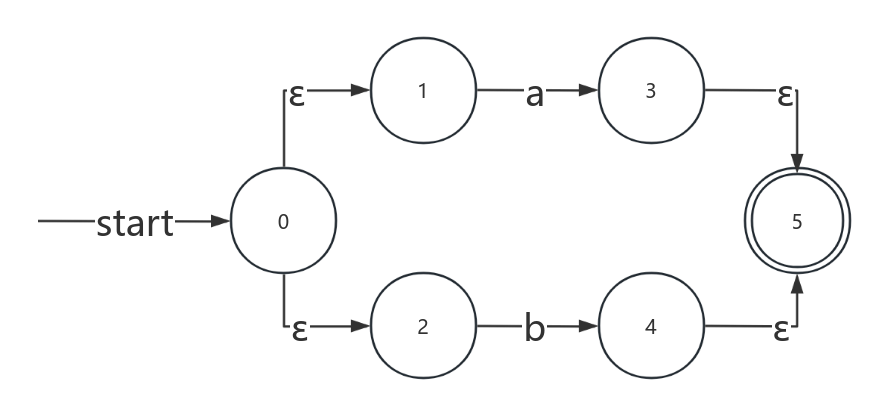
2.基于 Tompson 算法的 RE 转 NFA、基于子集法的NFA转换DFA和基于分割法的 DFA 最小化。Tompson转化方式的五种情况如图一至图四。

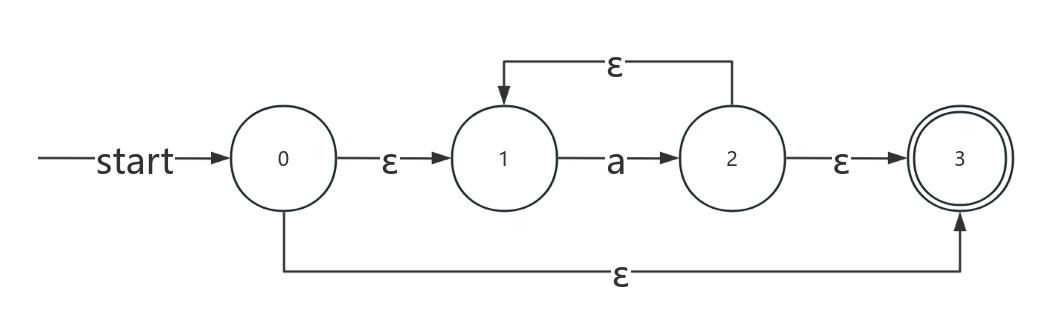
**图1.1 制造基础NFA（字符a） 图1.2 制造基础NFA（字符b）**



**图2 两个NFA的拼接（操作符.）**



**图3 两个NFA的合并（操作符|）**



**图4 闭包的NFA（操作符\*）**

1. **软件设计方法的选择**

**表3 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4 创建的模型**

|  |  |
| --- | --- |
| **分析模型** | **设计模型** |
| 系统流程图、用例图 | 数据结构图、模块结构图 |

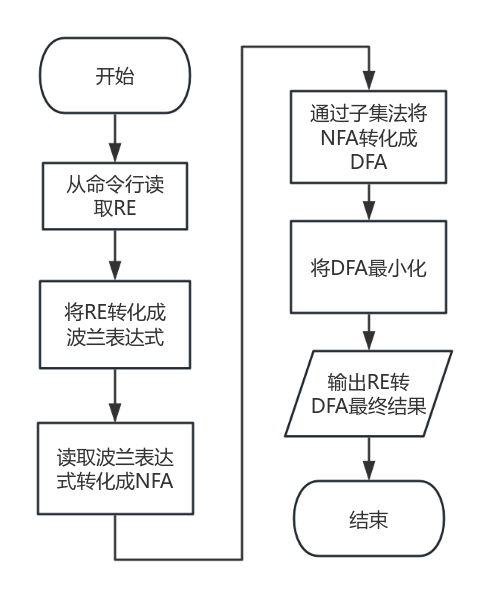
1. **分析模型**

**系统流程图：**

①基于自动分析机的词法分析器

程序首先读取源代码文件并通过deal函数对其进行预处理，去除注释和多余空格。接着调用lexerAnalyse函数逐词解析代码字符串，将其分解为词法单元（Word）。在lexerAnalyse 中，通过逐字符扫描判断当前单词的类型：若以数字开头，识别为数字；若以字母或下划线开头，判断是否为关键字或标识符；对其他字符或组合，则检查是否为合法符号。无法匹配的内容标记为未识别项。最终，将识别出的Word按类型存储并返回，用于后续的分析和输出。

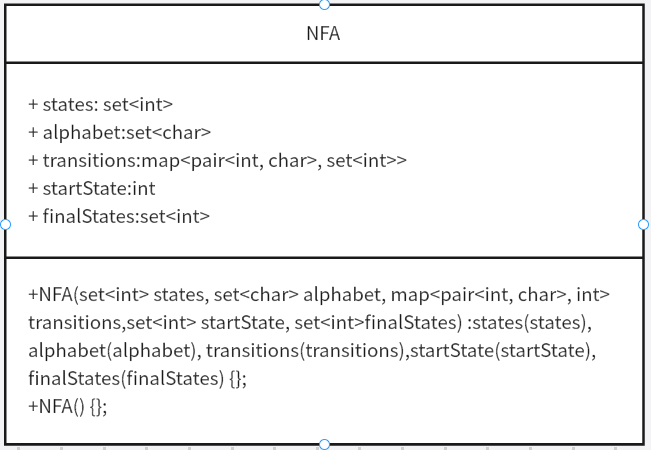
**图5 词法分析器**



②RE转DFA

首先解析输入的正则表达式，将之转化为电脑处理方便的波兰表达式，并通过Thompson构造法将其转换为一个非确定有限自动机（NFA），这包括处理连接、选择、闭包等操作生成相应的NFA片段。然后，使用子集构造法将NFA转化为DFA，其中每个DFA状态是NFA状态集合的表示，逐步处理所有转移关系直到完整生成。接着，对生成的DFA应用最小化算法，通过等价类划分合并冗余状态，从而得到最小化的DFA。最终输出包含状态集合、转移函数、初始状态和接受状态的DFA模型，用于高效的模式匹配和词法分析。

**图6 RE转DFA**

1. **设计模型**

**主要数据结构图：**

**（1）NFA：**

与 NFA 相关的函数包括：

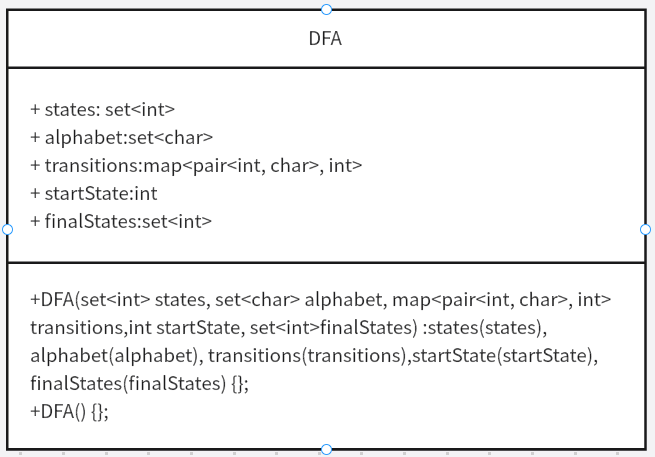
1. printNFA：打印 NFA 的状态集、输入字母表、状态转移函数、初始状态和终止状态集等信息。
2. createBasicNFA：构造一个简单的 NFA，用于表示一个字符的基本 NFA。
3. concatNFA：将两个 NFA 连接起来，生成它们的串联 NFA。
4. unionNFA：将两个 NFA 合并，生成它们的并集 NFA。
5. RE\_closure：对 NFA 应用 Kleene 星操作 **图7 NFA**

，生成一个新的 NFA。

1. readNFA：从文件读取 NFA 的定义，构造并返回相应的 NFA 对象。

这些函数共同作用于 NFA 的构造、操作和打印等任务。

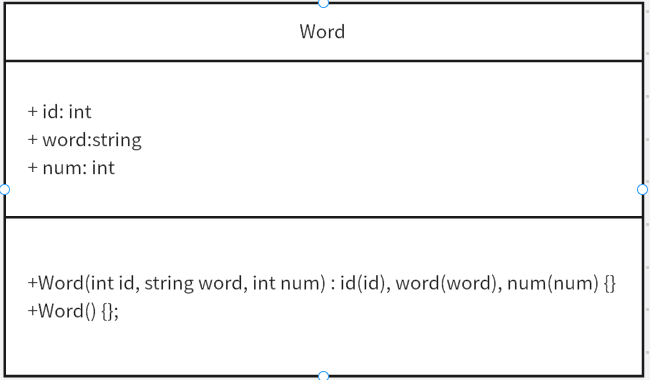
**（2）DFA:**

与 DFA 相关的函数包括：

1. epsilonClosure：用于计算 NFA 中状态集合的ε闭包。
2. move：在给定状态集合和输入符号下，计算可能的目标状态集合。
3. turnToDFA：实现从 NFA 到 DFA 的转换，包含确定化检查和子集构造法。
4. printDFA：打印DFA的状态集、输入字母表、转移函数、初始状态和终止状态等信息。
5. minimizedDFA：通过等价状态划分对 DFA 进行最小化，生成简化后的等效DFA。

这些函数协同作用，实现了从 NFA 转换为最小化 DFA 的完整流程。

**图8 DFA**

**（3）Word:**

与 Word 相关的函数包括：

1. deal：移除注释和多余空格，准备干净的输入字符串。
2. lexerAnalyse：将输入字符串分解为词法单元并生成 Word 对象列表。
3. printWords：根据Word的编号和类别输出分析结果。

这些与Word相关的函数协同作用，实现了对输入代码的词法分析，生成词法单元并分类处理的完整流程。

**图9 Word**

1. **主要算法描述**

**（1）RE转NFA（如下图10）:**

首先将RE表达式转换为后缀正则表达式，然后通过栈结构实现将后缀正则表达式转换为非确定性有限自动机（NFA）。后者遍历后缀表达式中的每个字符，遇到字母或数字时生成基本的 NFA，遇到操作符如 Kleene 星 (\*)、连接操作 (.) 或并集操作 (|) 时，分别应用相应的 NFA 操作（如闭包、连接、并集）生成新的 NFA，并将结果推入栈中。最终，栈中剩下的唯一NFA就是表示整个正则表达式的NFA。

**（2）NFA转DFA（如下图11）：**

NFA转换为DFA的过程是通过子集构造法实现的，首先需要检查给定的 NFA 是否已经是确定化的。如果NFA已确定化（即每个状态在每个输入符号下最多有一个目标状态），可以直接将其映射为DFA，无需进一步处理。如果不是，则开始子集构造法的核心步骤。初始状态是NFA的起始状态的ε-闭包，它被当作一个新的状态加入DFA的状态集中。

随后，通过队列逐步处理每个状态集合，对于每个集合和输入符号，计算该集合在符号作用下的目标状态集合，再对目标状态集合进行ε-闭包操作，得到新的状态。如果新状态尚未出现在 DFA 的状态集中，则将其加入队列和状态集中，同时更新 DFA 的状态转移函数。在这个过程中，函数会持续检查状态集合中是否包含 NFA 的终止状态，如果包含，则将其标记为 DFA 的终止状态。

为了简化表示，最终将每个状态集合映射为一个整数，对 DFA 的状态集、转移函数、初始状态和终止状态集进行扁平化处理，形成最终的DFA。这一方法通过合并 NFA 的所有可能路径，生成了一个行为等价但更加规范化的DFA，能够唯一确定每一个输入串的接受状态。

**（3）DFA最小化（如下图12）：**

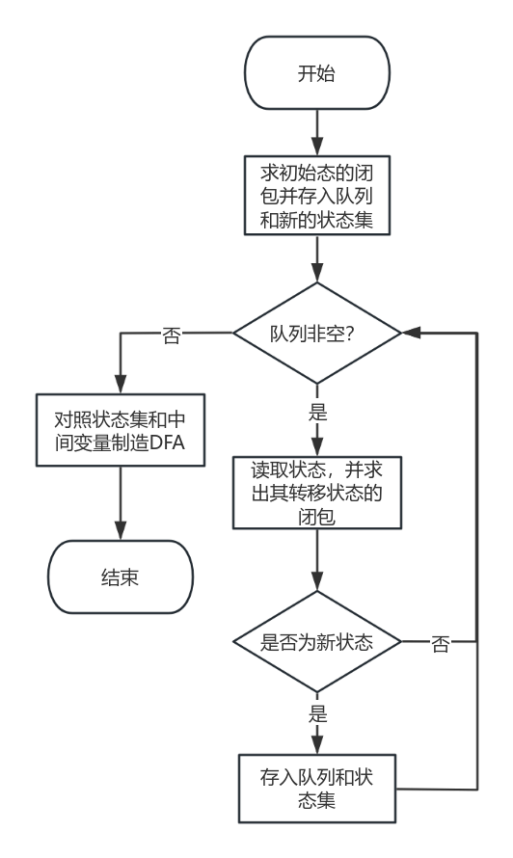
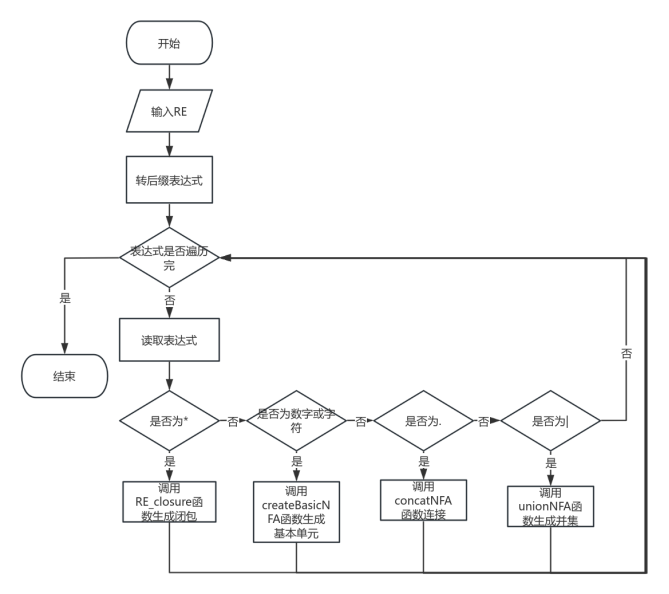
DFA的最小化过程是通过状态等价类划分实现的，首先将DFA的状态分为两组：终止状态和非终止状态，作为初始分组。接下来，针对每个分组中的状态，按照输入符号的转移目标进行细化。如果同一分组中的状态在某一输入下到达了不同的目标分组，它们会被进一步分裂到不同的新分组中。这个过程是一个迭代过程，每次更新分组后都会检查是否需要继续细化，直到分组不再变化为止，此时所有状态已经被划分为最小的等价类。

在完成分组划分后，将每个分组映射为一个新的 DFA 状态，并重新构造 DFA 的状态集、转移函数、初始状态和终止状态。新的 DFA 状态由分组的编号表示，转移函数根据旧 DFA 的转移和分组的映射关系生成。最终，所有等价的状态都被合并为一个状态，生成的 DFA 状态数最小化，同时保持了原 DFA 的语言识别能力。这个过程充分利用了状态等价性和分组细化的思想，是 DFA 最小化的标准实现方法。

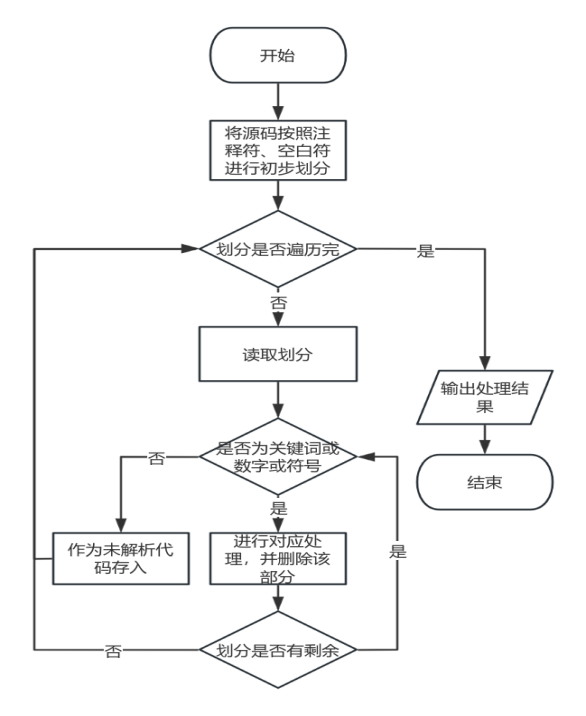
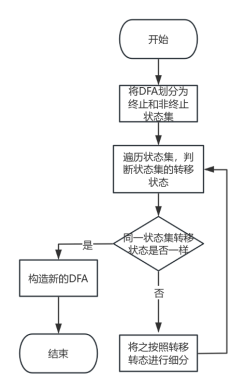
**（4）词法分析器（如下图13）：**

词法分析器的逻辑是将输入的字符串分解成一个个有意义的词法单元（Word），即“词”。首先，将输入字符串按空格分隔成单独的词，然后逐个处理每个词。对于每个词，依次检查每个字符。对于关键字和标识符，通过检查字符是否符合字母或数字（以及是否包含下划线）来确定。遇到关键字、标识符或数字时，将其映射为对应的词法单元类型，从次中剔除，并将其加入结果中。其中，处理符号时先检查两个字符的组合符号，如果符合规则就生成相应的词法单元，否则进行单个符号的判断并处理。

最终，词法分析器会根据所有规则将字符串中的各个部分转换为一系列的词法单元，并返回这些词法单元的列表，以便后续的语法分析和其他编译过程。



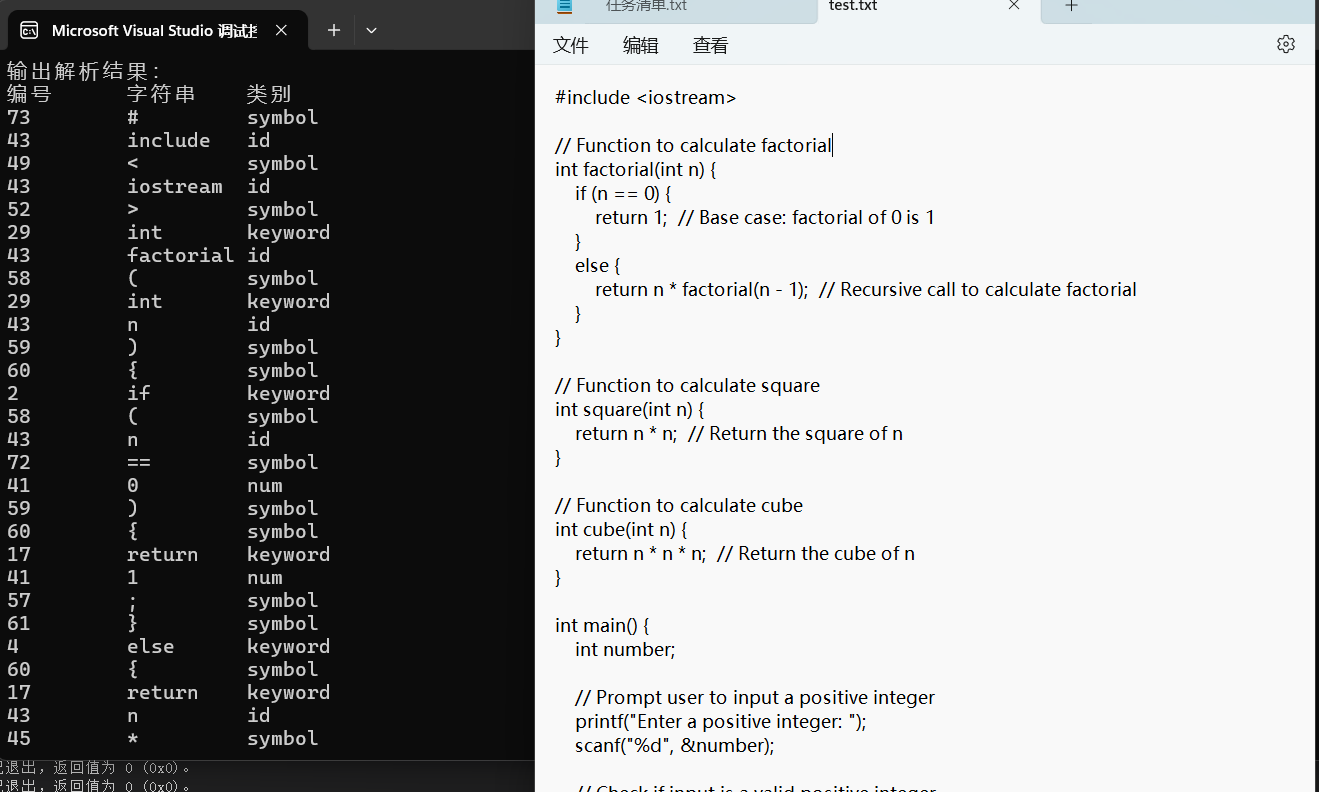
**图10 RE转NFA 图11 NFA转DFA**



**图12 DFA最小化 图13 词法分析器**

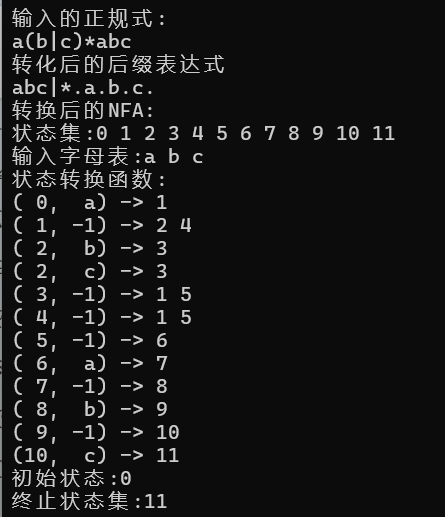
1. **测试数据与测试效果**

**（1）词法分析器：**

****

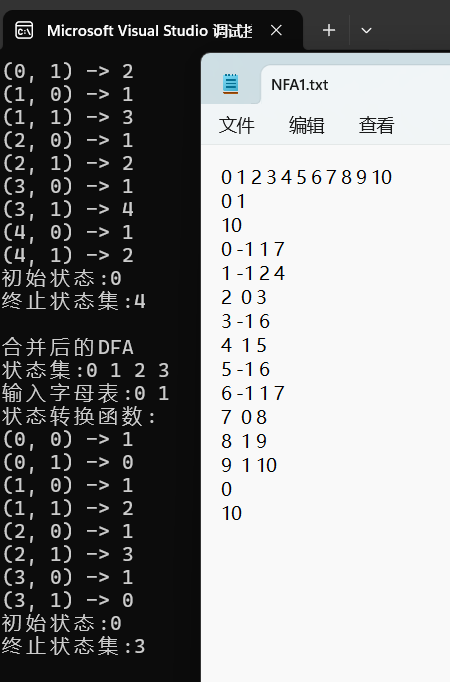
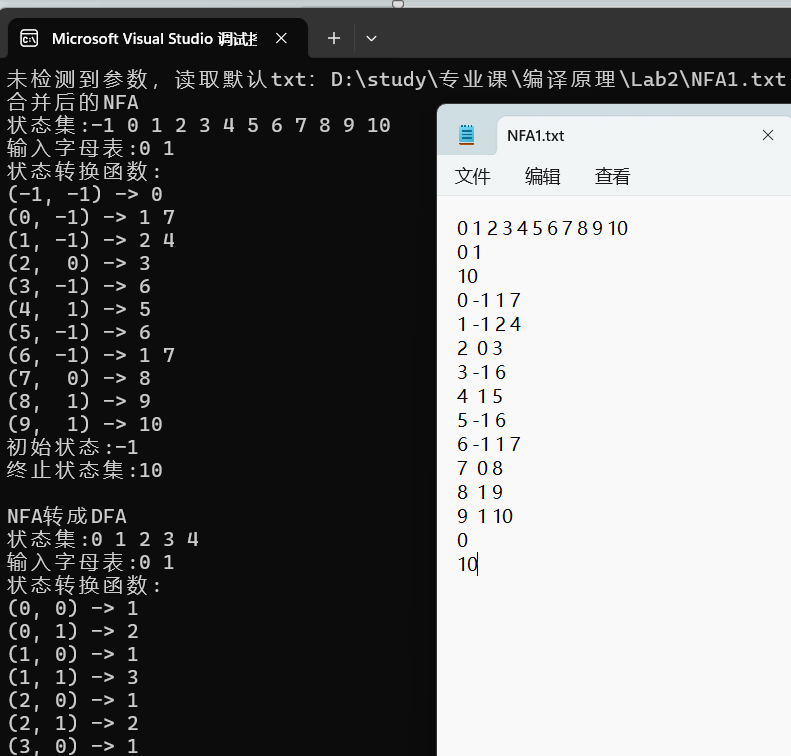
**图14 词法分析器运行结果**

**（2）RE转NFA:a(b|c)\*abc**



**图15 RE转NFA运行结果**

**（3）NFA转DFA：**



**图16、17 NFA转DFA运行结果**

1. **实验总结**

在本次实验中，我实现了正则表达式（RE）到确定性有限自动机（DFA）的转换，设计了一个简单的词法分析器，并使用Lex工具生成了词法分析器。首先，我通过将正则表达式转换为NFA，再通过子集构造法将其转为DFA，成功实现了自动机的转换过程。在此过程中，我深入理解了正则表达式和自动机之间的关系，并掌握了如何通过状态集合来优化DFA。

在词法分析器的实现中，我设计了一个能够识别关键字、变量名、数字和符号的分析器，利用有限状态机逐步分析输入字符串，并输出相应的词法单元。通过手动编写分析器代码，我加深了对词法分析的理解。

此外，使用Lex工具生成词法分析器，简化了实际开发中的词法分析过程。我学会了如何通过Lex定义词法规则并自动生成相应的分析器，使得开发变得更加高效。

本次实验加深了我对编译原理中正则表达式、自动机以及词法分析的理解，提升了我的编程能力，尤其是在自动机的实现和Lex工具的使用方面。