**《形式语言与自动机》**

**课程实验报告**



NFA到DFA的转化

**小组成员：**

20129211309班 2019211404号 戎惠强 成员分工：文档

20129211309班 2019211403号 刘明哲 成员分工：代码

20129211309班 2019211409号 崔光程 成员分工：测试

1. **实验目的**

编程实现NFA到DFA的转化，理解不同自动机的转化过程

1. **实验内容**

有限状态自动机是描述控制过程有力工具。有限状态自动机有不同的类型，例如，确定有限状态自动机（DFA）和不确定有限状态自动机（NFA）。这些不同类型的自动机之间可以等价转化。我们在实际应用中，可以利用某种类型的自动机更加方便刻画实际系统，然后再利用等价转化算法实现不同类型的自动机转化。

本实验要求编程实现NFA到DFA的自动转化。输入自己设定的不确定有限

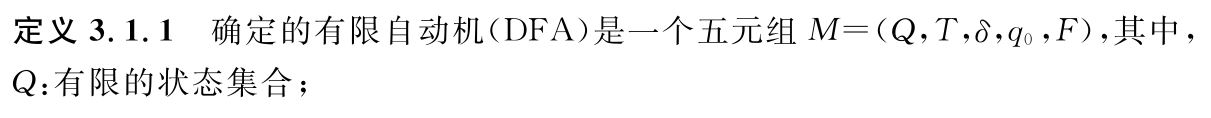
自动机描述格式，输出对应的确定有限自动机。

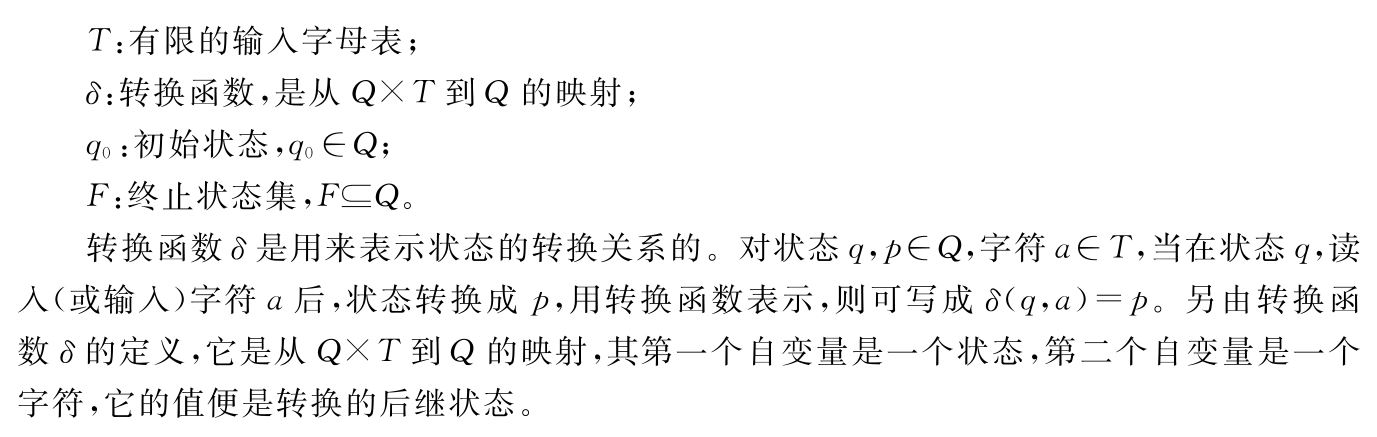
1. **实验环境**

Windows10操作系统，使用VS2017开发环境，C语言编写。

1. **实验原理**

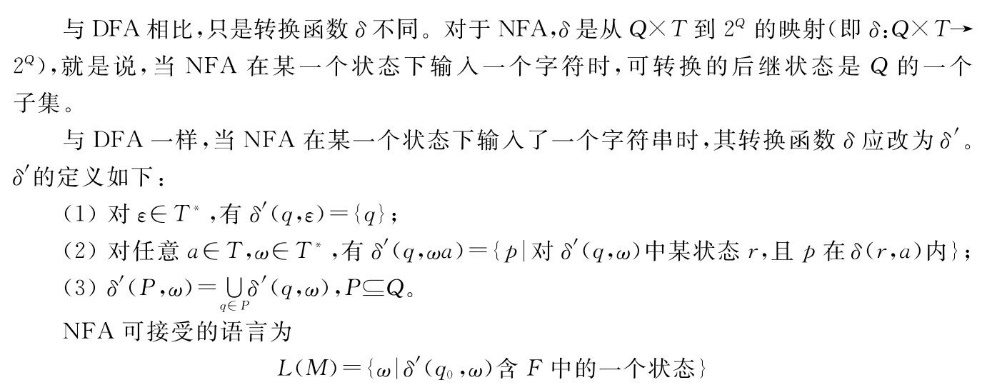
**（1）DFA**



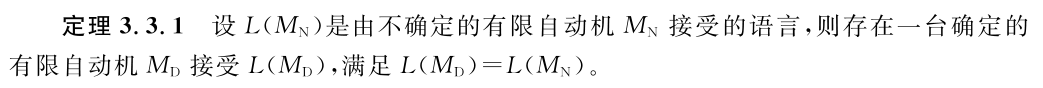


**（2）NFA**



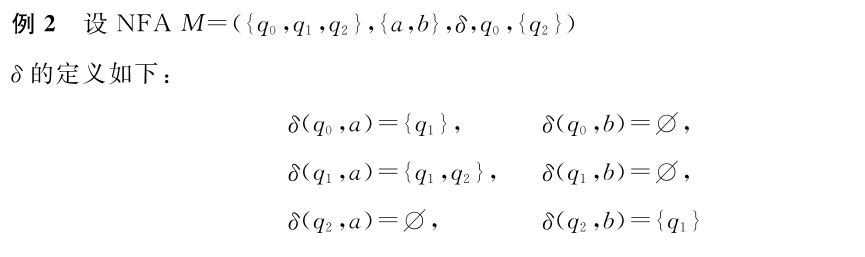


**（3）NFA与DFA等价**

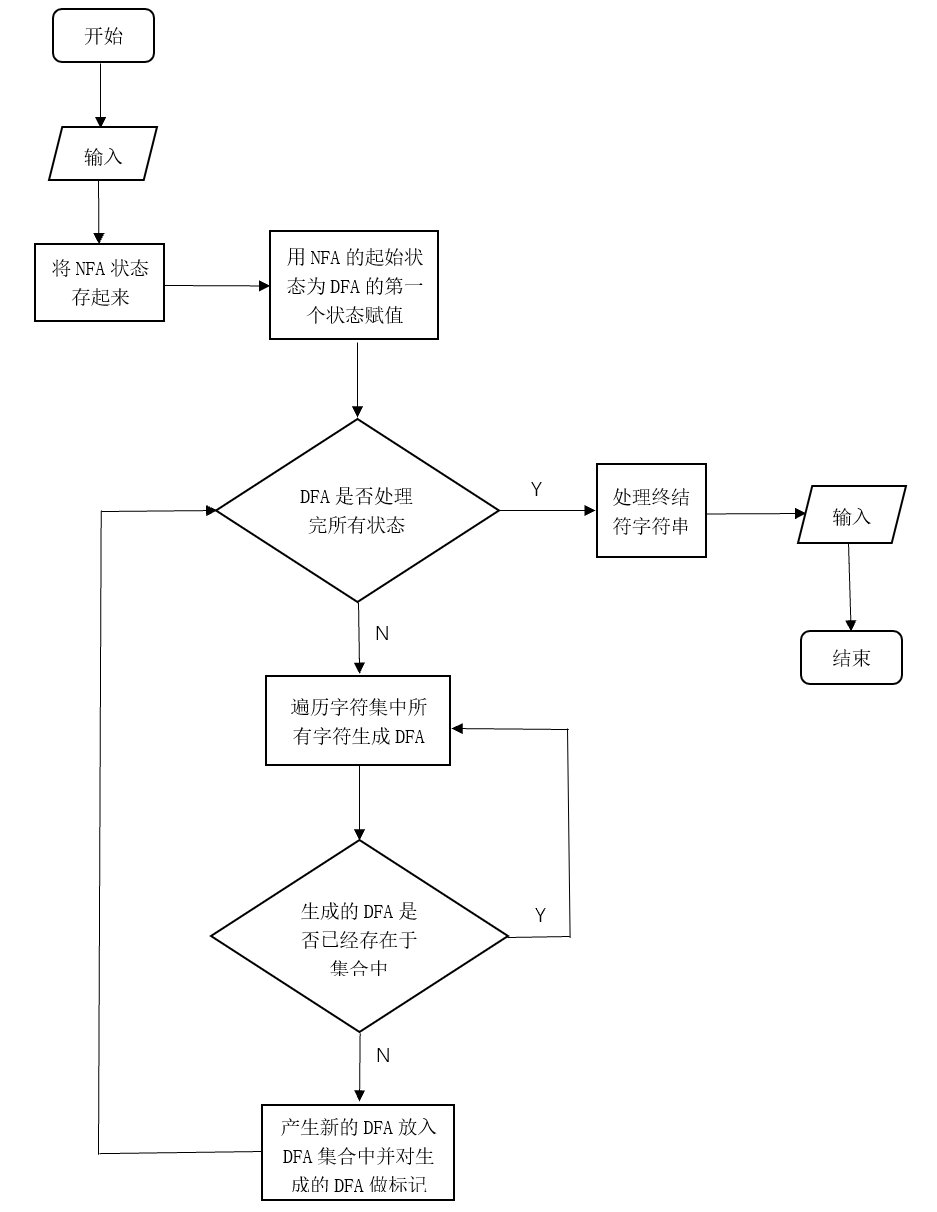


证明略

**（4）子集构造法举例**

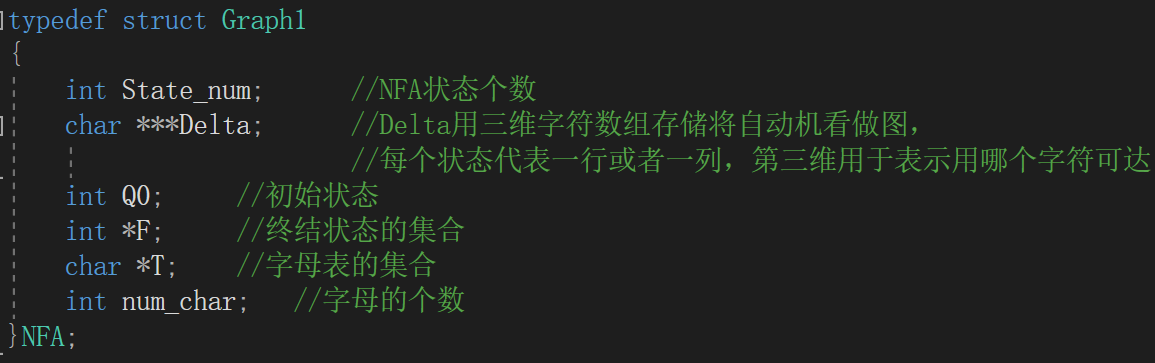


1. **程序基本模块**
2. 程序流程图

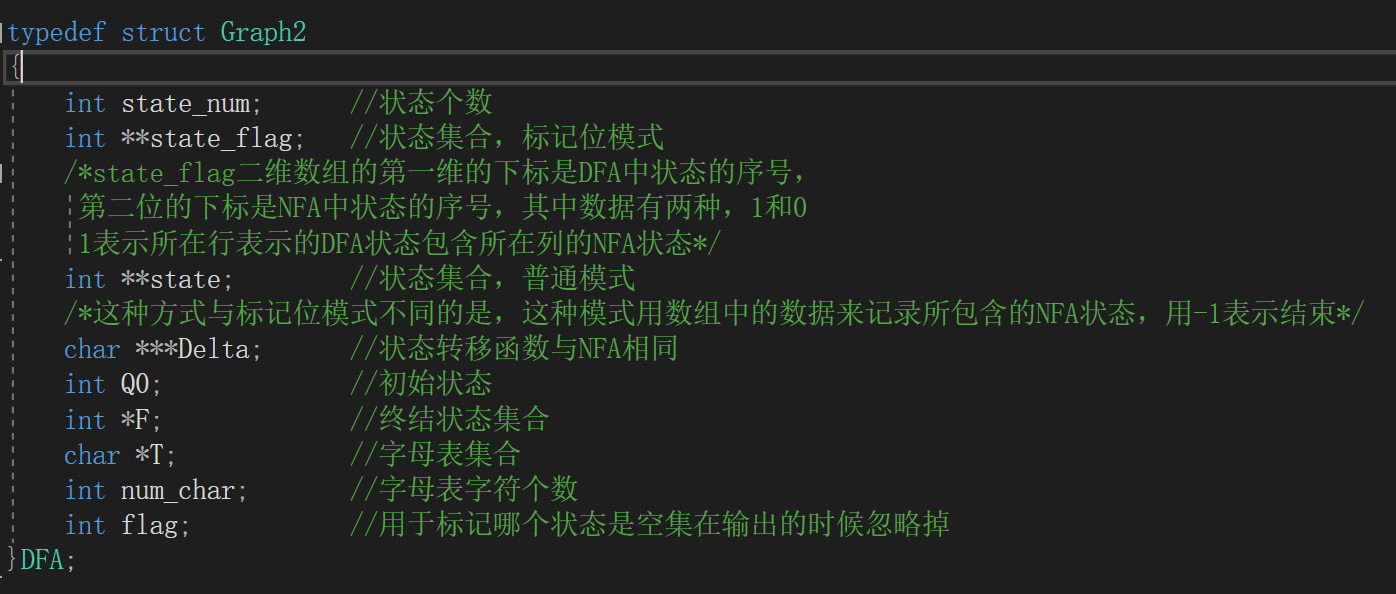


1. 数据结构定义

//NFA存储结构：



//DFA存储结构：



常量定义：

#define MAX\_NAME\_LEN 4 //状态名最长不超过四个字符

1. 各函数定义及意义

void build\_NFA(NFA \*m, char \*name[MAX\_NAME\_LEN + 1]); //读入NFA并处理数据生成一个NFA

void get\_name(char \*name); //用于读取状态名

int match\_name(int num, char \*name[MAX\_NAME\_LEN + 1], char \*state); //用于匹配状态名，匹配到返回状态序号，没有匹配到则返回-1

void init\_NFA(int num, NFA \*m); //初始化NFA，为NFA分配空间

void invert(NFA \*m1, DFA \*m2); //转化函数的声明

int match\_DFA(int num\_DFA, int num, int \*\*state, int \*set); //匹配DFA的状态，找到了返回状态序号，没有找到返回-1

void init\_DFA(int num, DFA \*n);//DFA的初始化

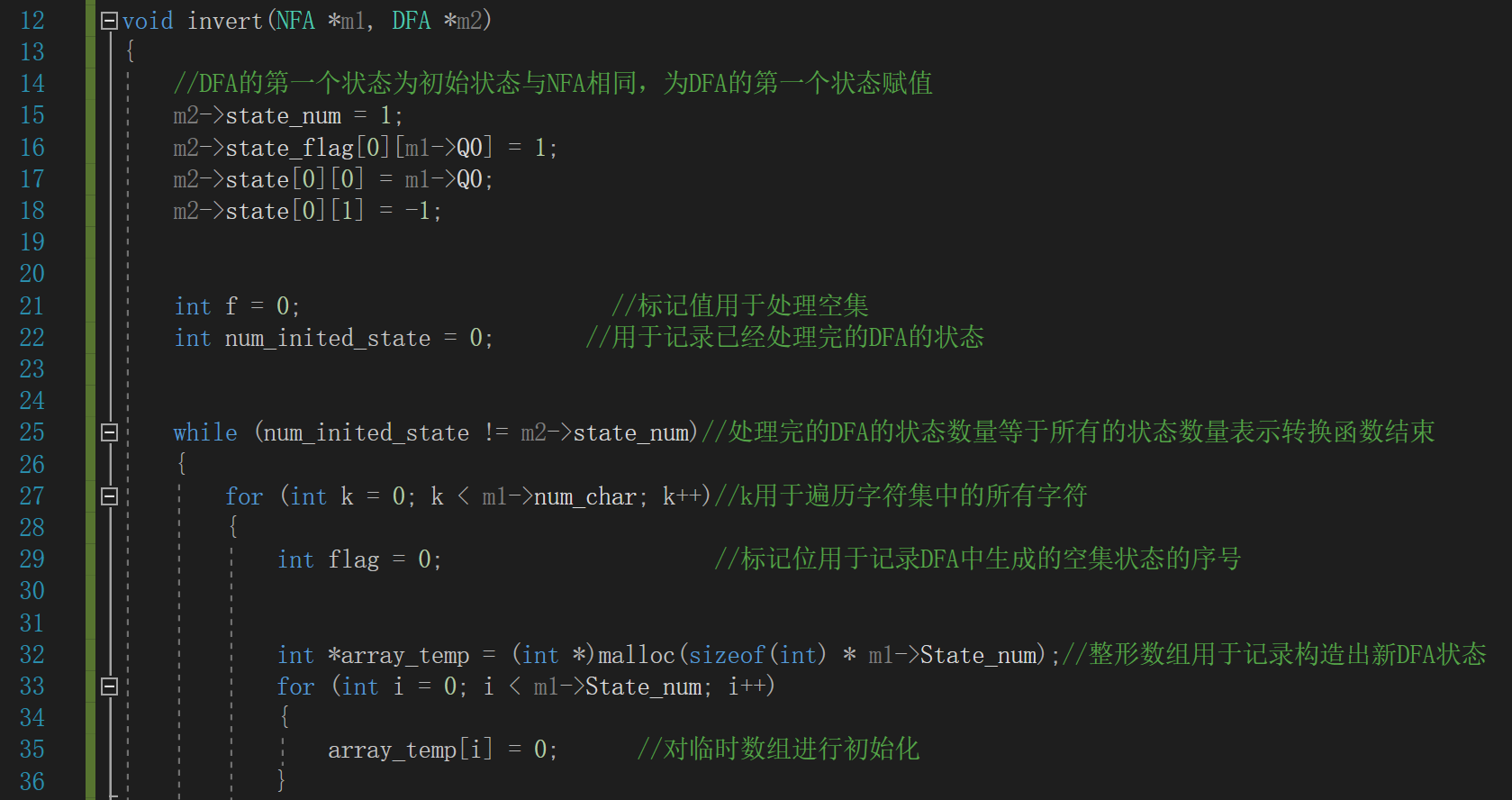
int Pow(int x, int y); //幂函数计算

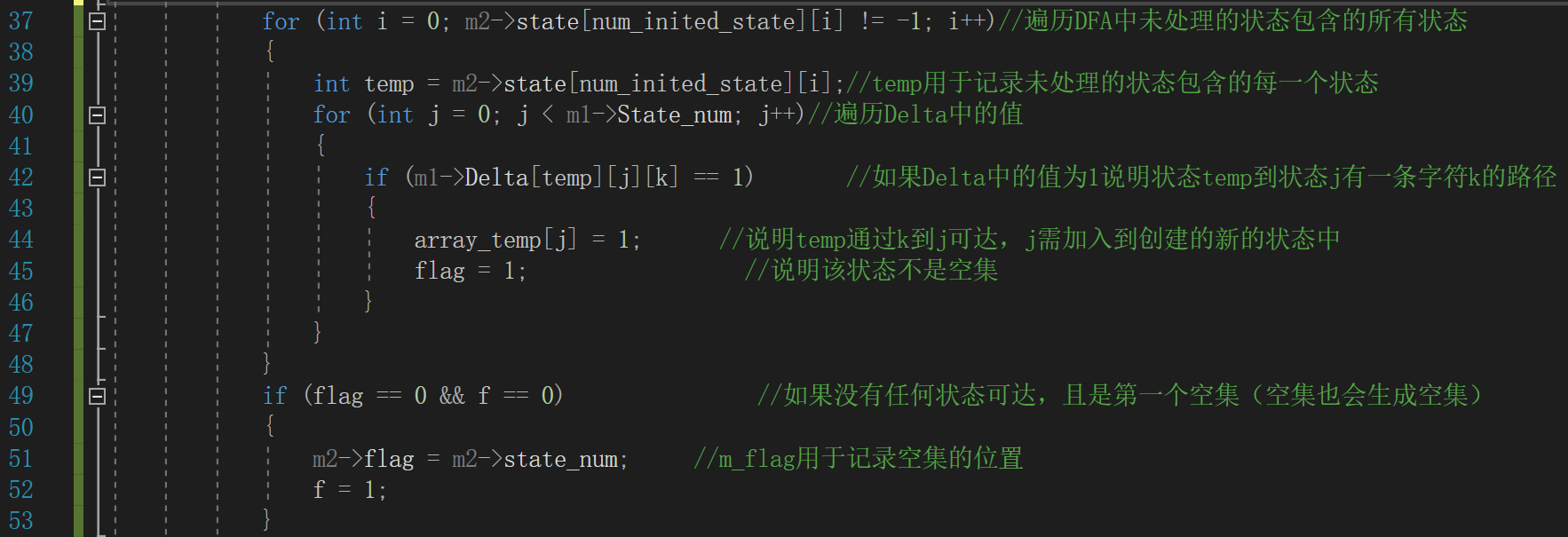
void Print(const DFA \*m, const char \*name[MAX\_NAME\_LEN]);//输出DFA

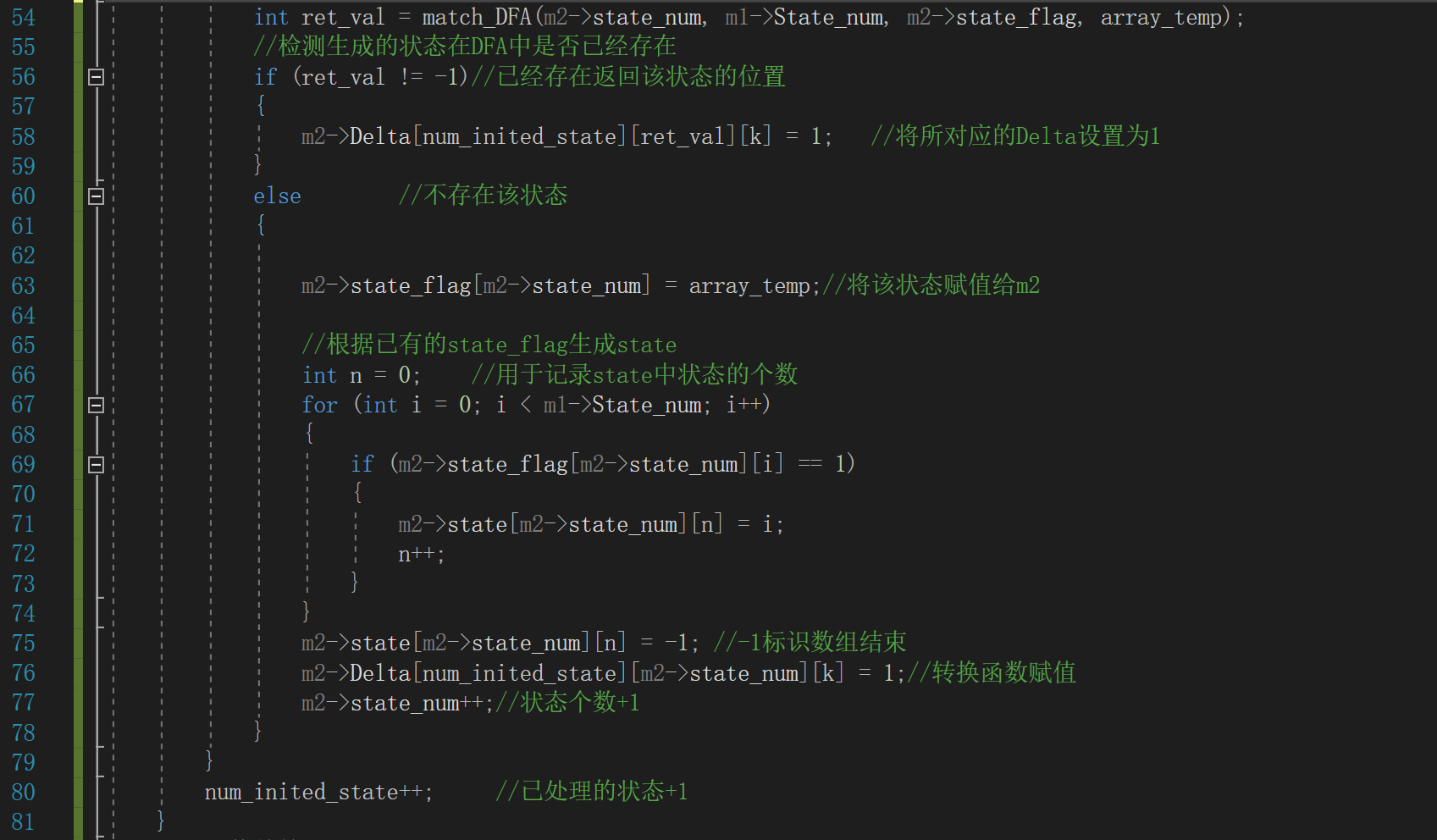
1. 主函数



5、核心算法

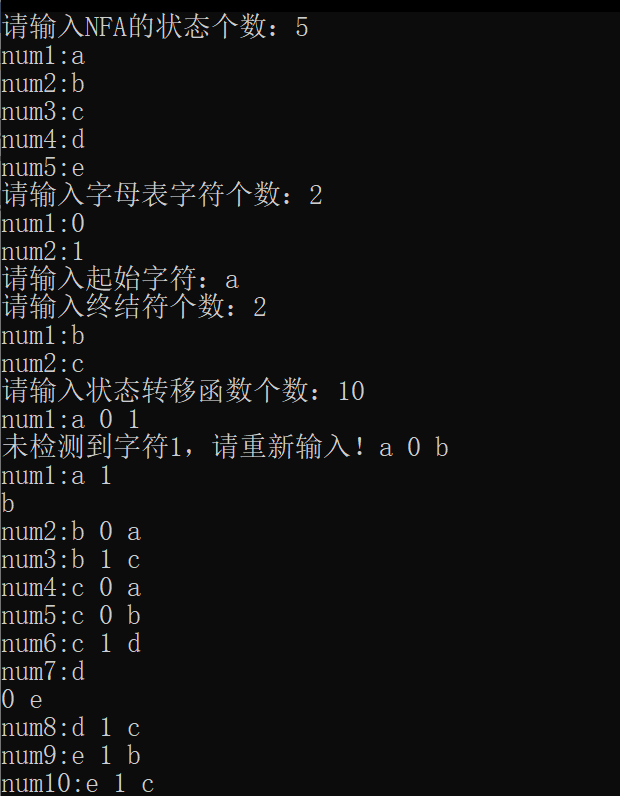




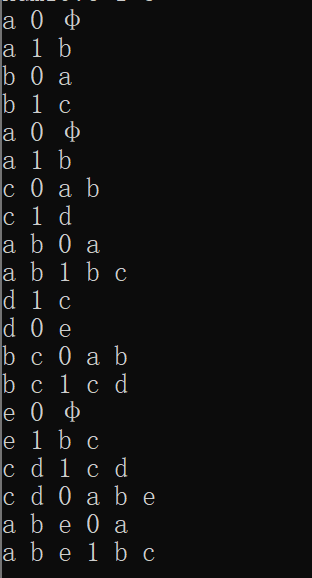


1. **测试用例**

**NFA=({a,b,c,d,e},{0,1},δ,{a},{b,c})**



**输出：**



1. **程序设计时的问题及解决方案**

最初使用的是邻接矩阵存储，使用二维字符数组来对应转换关系，但是这样的话不符合多图结构，两个状态之间如果有多种转换关系，只会剩余最后输入的字母。所以经过考虑之后，我们使用了三维数组来进行存储，成功解决问题。

刚开始对于转换方式的核心代码这一块，我们考虑了两种方式，根据状态来进行遍历以及根据字母表来进行遍历，在尝试根据状态遍历时，对于重复状态的判定以及对后续状态的再转换问题上我们遇到了麻烦，最后重新决定使用遍历字母表的方式进行，使用另外一个结构来存储新表，也顺利解决问题。

1. **实验总结**

这次实验我们完成了NFA到DFA的转化，对自动机有了更加深入的了解，同时在写代码的同时，也对曾经学过的图等数据结构进行了复习与巩固，在解决代码问题方面也有了进步，而对于小组合作模式也有了更充分的了解，我们彼此的交流和合作也更好了。