实验2.2 构造与下推自动机等价的上下文无关文法

## 小组成员

班级：2022211307 陈晨 学号：2022211259 分工：（实验2.2）代码其他部分，程序调试与分析，实验报告程序调试部分

班级：20222211307 于孟孟 学号：2022211260 分工：（实验2.2）代码PDA向CFG转换部分，实验报告其他部分

## 实验目的

编程实现由下推自动机构造等价的上下文无关文法的算法。

## 实验内容及环境

2.1 实验内容

下推自动机和上下文无关文法，是用于描述上下文无关语言的两种方式。对于给定的一个下推自动机，必存在一个上下文无关文法，使得该文法产生的语言与下推自动机接受的语言等价。

要求

（1）编程实现由下推自动机构造等价的上下文无关文法的算法。输入为一个下推自动机，输出为与该下推自动机等价的上下文无关文法。

（2）将输出的等价上下文无关文法作为输入，利用所实现的上下文无关文法变换算法，输出与该文法等价的没有ε产生式、单产生式、无用符号的上下文无关文法。

至少使用如下的下推自动机进行程序的正确性验证。

设PDA M＝（{q0,q1},{a,b},{B,z0},δ,q0,z0,Φ）

δ定义为：δ（q0,b,z0）={( q0,Bz0)}

δ（q0,b,B）={( q0,BB)}

δ（q0,a,B）={( q1,ε)}

δ（q1,a,B）={( q1,ε)}

δ（q1,ε,B）={( q1,ε)}

δ（q1,ε,z0）={( q1,ε)}

2.2 实验环境

利用vscode软件，利用C++进行代码编写

## 概要设计

### 3.1 功能设计

（1）接收用户输入的七元组形式定义的空栈接受的PDA并存储

（2）构造与PDA等价的CFG并输出

### 3.2 数据结构

#### （1）PDA

#### （2）CFG

#### （3）PDA的转换函数

#### （4）为CFG新定义的文法的非终结符

#### （5）新定义的CFG的产生式

### 3.3 核心算法

#### （1）PDA的输入

由class PDA的构造函数完成；

用户根据文字提示依次输入PDA的状态数目、PDA的状态集，PDA的字符个数、PDA的字母表，PDA的栈符号个数、PDA的栈符号集合，PDA的转移函数的个数、PDA的转移函数，PDA的起始状态，PDA的栈起始符，PDA的终止状态数目、PDA的终止状态集（空栈接受的PDA终止状态数目为0）；

构造函数读取PDA的信息，构造PDA，以class PDA的形式存储

#### （2）打印PDA，向用户确认输入

为每个struct和class重载"<<"运算符(如3.2所示），由标准输出流cout输出

#### （3）PDA向CFG的转换

由class PDA的成员函数CFG PDA::generate\_cfg()完成；

完成的功能：

* 由CFG的字母表构建文法的终结符表
* 对于每一个状态r，将S->[q0, z0, r]添加至class CFG中由起始符开始的生成式vector中；将生成式右式添加至class CFG中的非终结符表(起始符S由CFG的构造函数添加，如3.2所示）
* 遍历每一个转换函数
  + 若转换函数右边的栈顶符号为空，则文法生成终结符(即PDA的输入字符）；构造对应的CFG产生式和非终结符，添加至class CFG对应部分的vector中(检查是否重复，不重复则添加) **即：若δ(q, a, z)含有(γ, ε)，则将[q, z, γ]->a加入到产生式中。**
  + 若转换函数右边的栈顶符号为单个字符，则文法生成一个终结符+一个非终结符；遍历状态集合，对所有状态序列构造非终结符(数据结构见3.2），添加至CFG非终结符列表；遍历构造的非终结符，对每一个非终结符构造转换函数，添加至CFG转换函数列表
  + 若转换函数右边的栈顶符号为两个字符，则文法生成一个终结符+两个非终结符；双重遍历状态集合，对所有状态序列构造非终结符，(数据结构见3.2），添加至CFG非终结符列表；遍历构造的非终结符，对每一个非终结符构造转换函数，添加至CFG转换函数列表
  + 若转换函数右边的栈顶符号为三个字符，则文法生成一个终结符+三个非终结符；三重遍历状态集合，对所有状态序列构造非终结符，(数据结构见3.2），添加至CFG非终结符列表；遍历构造的非终结符，对每一个非终结符构造转换函数，添加至CFG转换函数列表 **即：若δ(q, a, z)含有(γ, B1B2...Bk)，则对q中每一个状态序列**q1,q2,...qk，把形[q, z, qk]->a[γ, B1, q1][γ, B2, q2]...[qk-1, Bk, qk]加入到P中。
  + 未考虑更负责的情况

代码如下：

#### （4）打印构造完成的CFG

为每个struct和class重载"<<"运算符(如3.2所示），由标准输出流cout输出

#### （5）将终态接受的PDA转换为空栈接受的PDA

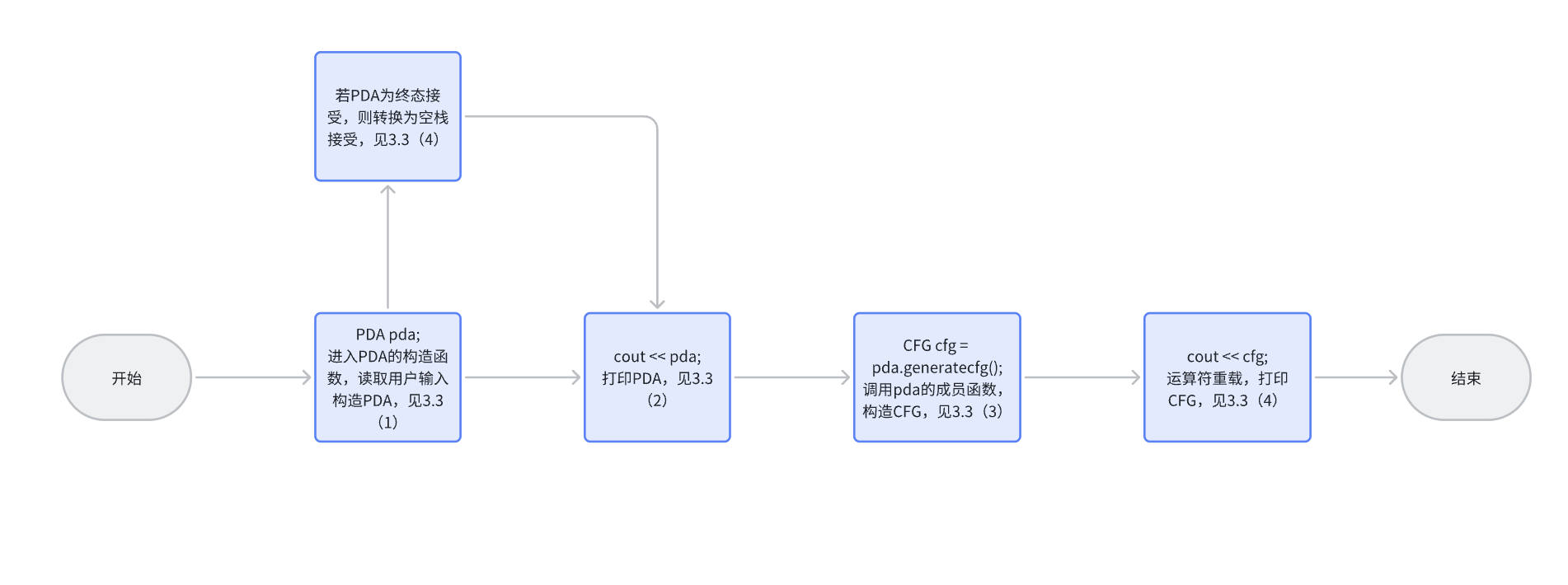
检测输入的PDA终态数目是否为0，若非零，则将其转换为空栈接受的PDA

代码如下

### 3.4 代码流程

#### main.cpp程序代码

#### 流程图



## 程序调试

#### （1）程序的输入、输出格式

**输入格式：**

例：

请输入PDA的状态数目：

**2**

请输入PDA的状态集（用空格分隔），以回车结束（如果您输入的是终态接受的PDA，请不要将qe和ql作为状态名称）：

**q0 q1**

请输入PDA字符的个数

**2**

请输入PDA字母表（用空格分隔），请确保您输入的字符不会互为前缀，以回车结束：（如果您输入的是终态接受的PDA，请不要将z1作为栈字符）

**a b**

请输入PDA的栈符号的个数：

**2**

请输入PDA的栈符号集合（用空格分隔），以回车结束：

**B z0**

请输入您即将输入的转移函数的个数:

**6**

请输入转移函数，格式为p(当前状态) q(输入字符) u(当前栈顶符号) r(下一个状态) v(输入后栈顶符号)，以回车结束每一个转移函数 ，请用[empty]表示空字符：

**q0 b z0 q0 Bz0**

**q0 b B q0 BB**

**q0 a B q1 [empty]**

**q1 a B q1 [empty]**

**q1 [empty] B q1 [empty]**

**q1 [empty] z0 q1 [empty]**

请输入初始状态：

**q0**

请输入栈起始符：

**z0**

请输入终止状态数目:

**0**

请输入终止状态集合，以空格分隔，以回车结束：

**输出格式：**

例：

状态集如下:

**q0 q1**

符号表如下:

**a b**

栈符号如下:

**B z0**

转换函数如下:

**(q0, b, z0)->(q0, Bz0)**

**(q0, b, B)->(q0, BB)**

**(q0, a, B)->(q1, [empty])**

**(q1, a, B)->(q1, [empty])**

**(q1, [empty], B)->(q1, [empty])**

**(q1, [empty], z0)->(q1, [empty])**

起始字符如下:

**q0**

栈起始符号如下:

**z0**

栈终结状态如下:

生成的CFG如下所示:

非终结符如下:

**[q0, z0, q0]**

**[q0, z0, q1]**

**[q0, B, q0]**

**[q0, B, q1]**

**[q1, B, q1]**

**[q1, z0, q1]**

终结符如下:

a b

文法产生式如下:

**S->[q0, z0, q0]**

***S->[q0, z0, q1]***

***[q0, z0, q0]->b[q0, B, q0][q0, z0, q0]***

***[q0, z0, q0]->b[q0, B, q1][q1, z0,* q0]**

**[q0, z0, q1]->b[q0, B, q0][q0, z0, q1]**

**[q0, z0, q1]->b[q0, B, q1][q1, z0, q1]**

**[q0, B, q0]->b[q0, B, q0][q0, B, q0]**

**[q0, B, q0]->b[q0, B, q1][q1, B, q0]**

**[q0, B, q1]->b[q0, B, q0][q0, B, q1]**

**[q0, B, q1]->b[q0, B, q1][q1, B, q1]**

**[q0, B, q1]->a**

**[q1, B, q1]->a**

**[q1, B, q1]->[empty]**

**[q1, z0, q1]->[empty]**

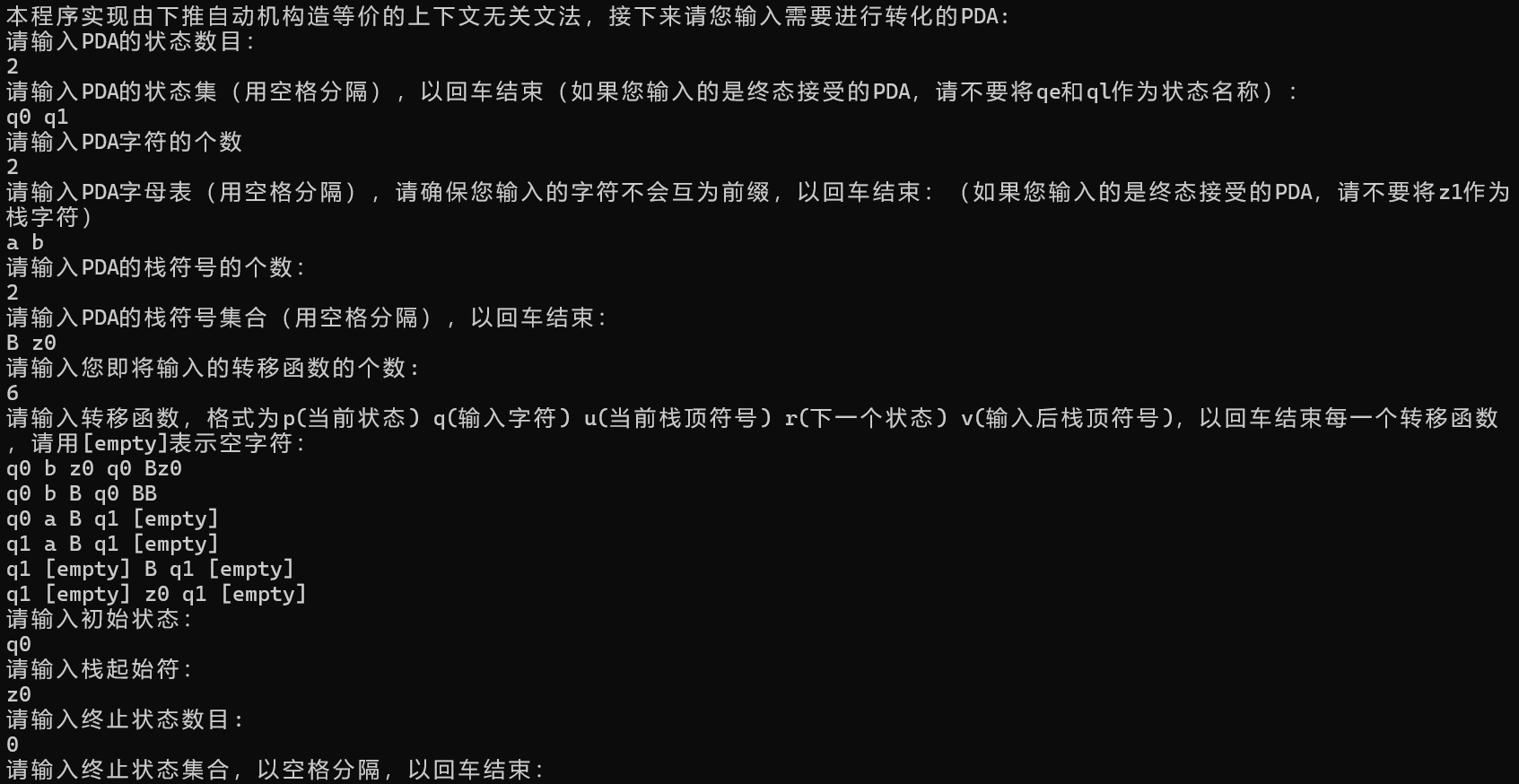
文法的起始符如下:

**S**

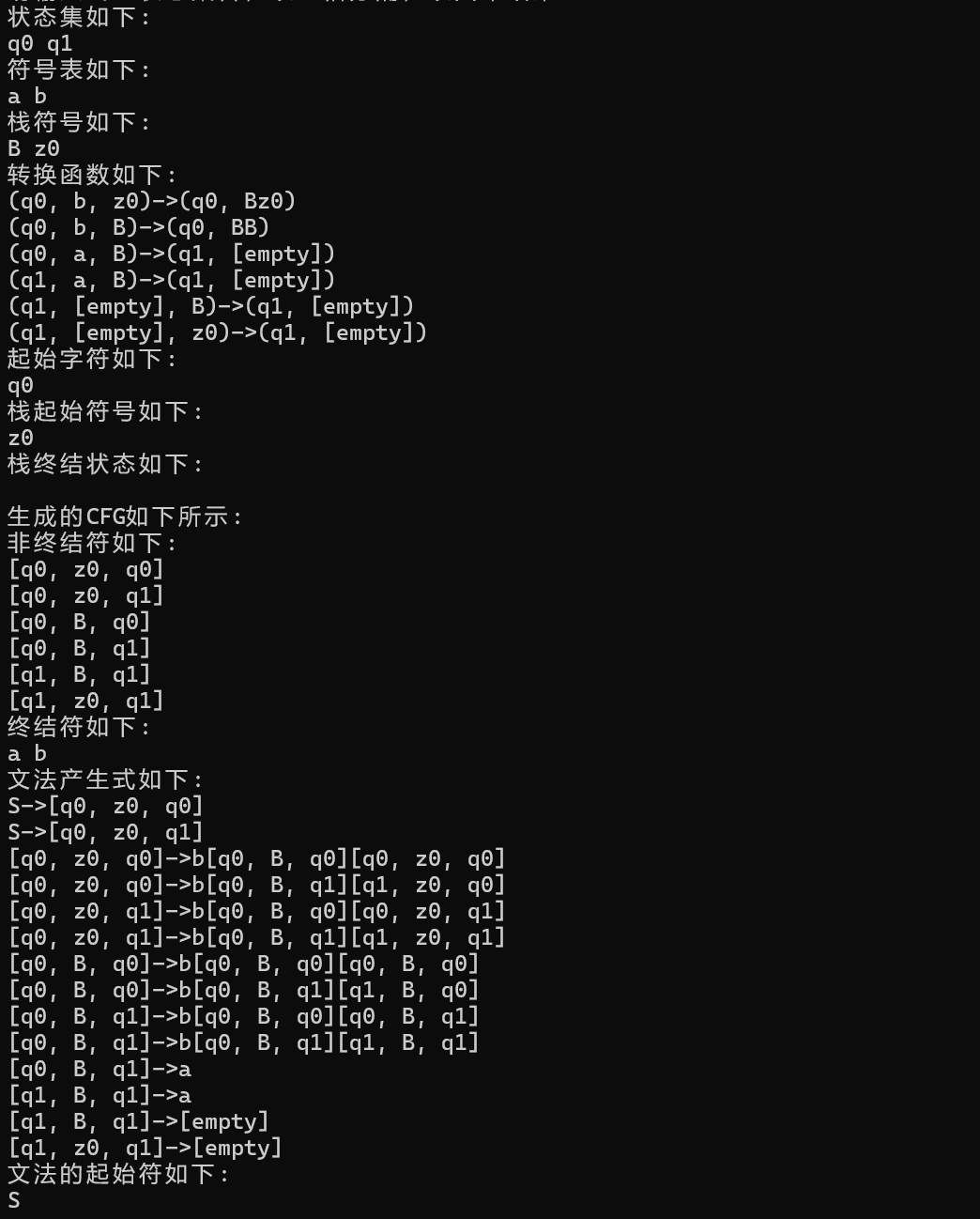
#### （2）程序的测试用例，输入、输出以及执行效果

**用例一：（实验给出）**

输入：

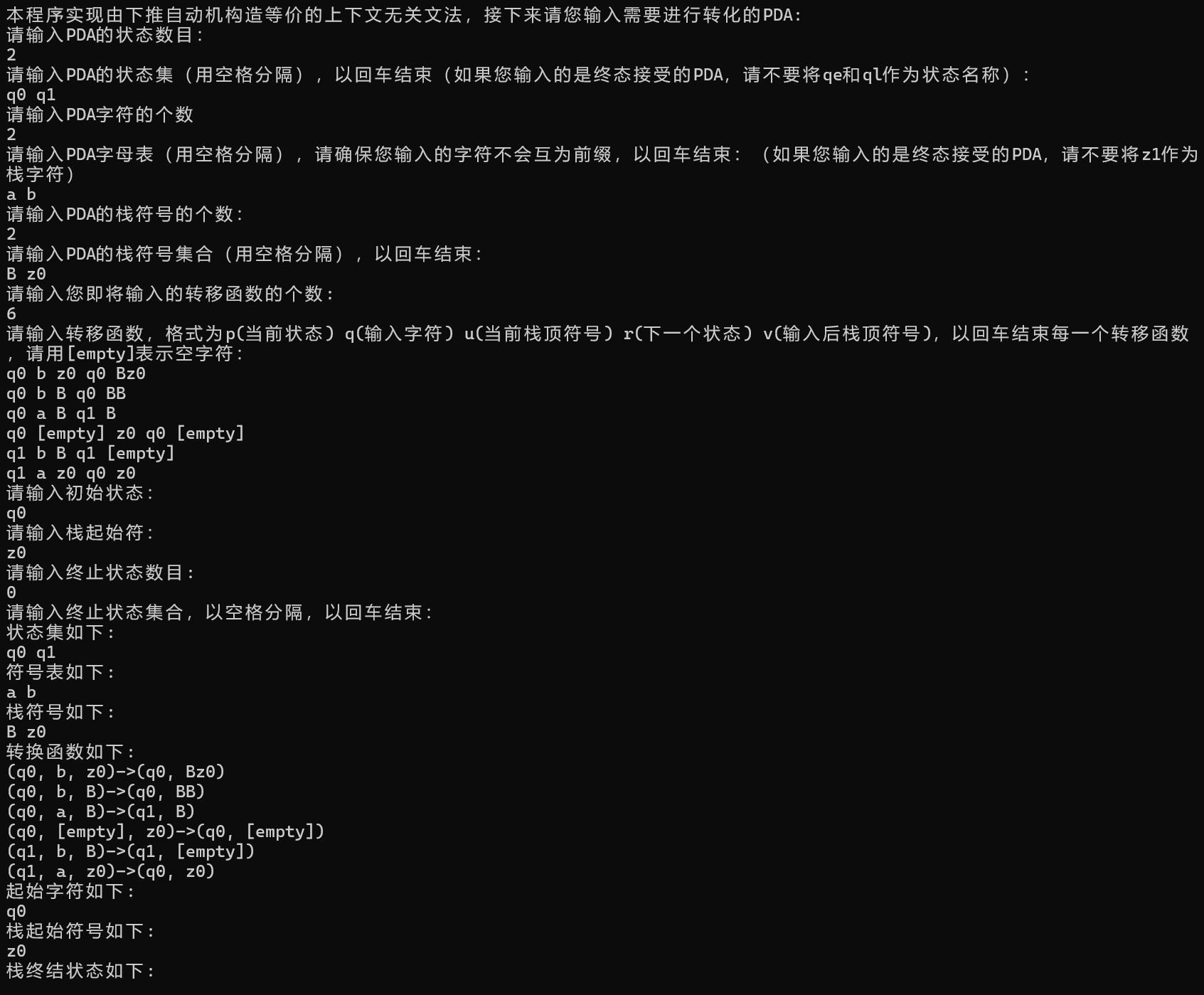


输出：

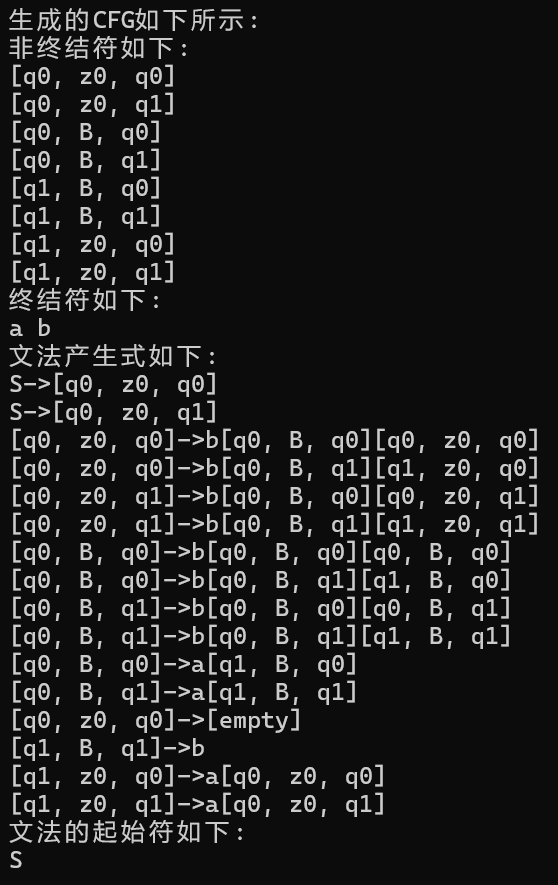


**用例二：（空栈接受的PDA）**

输入：

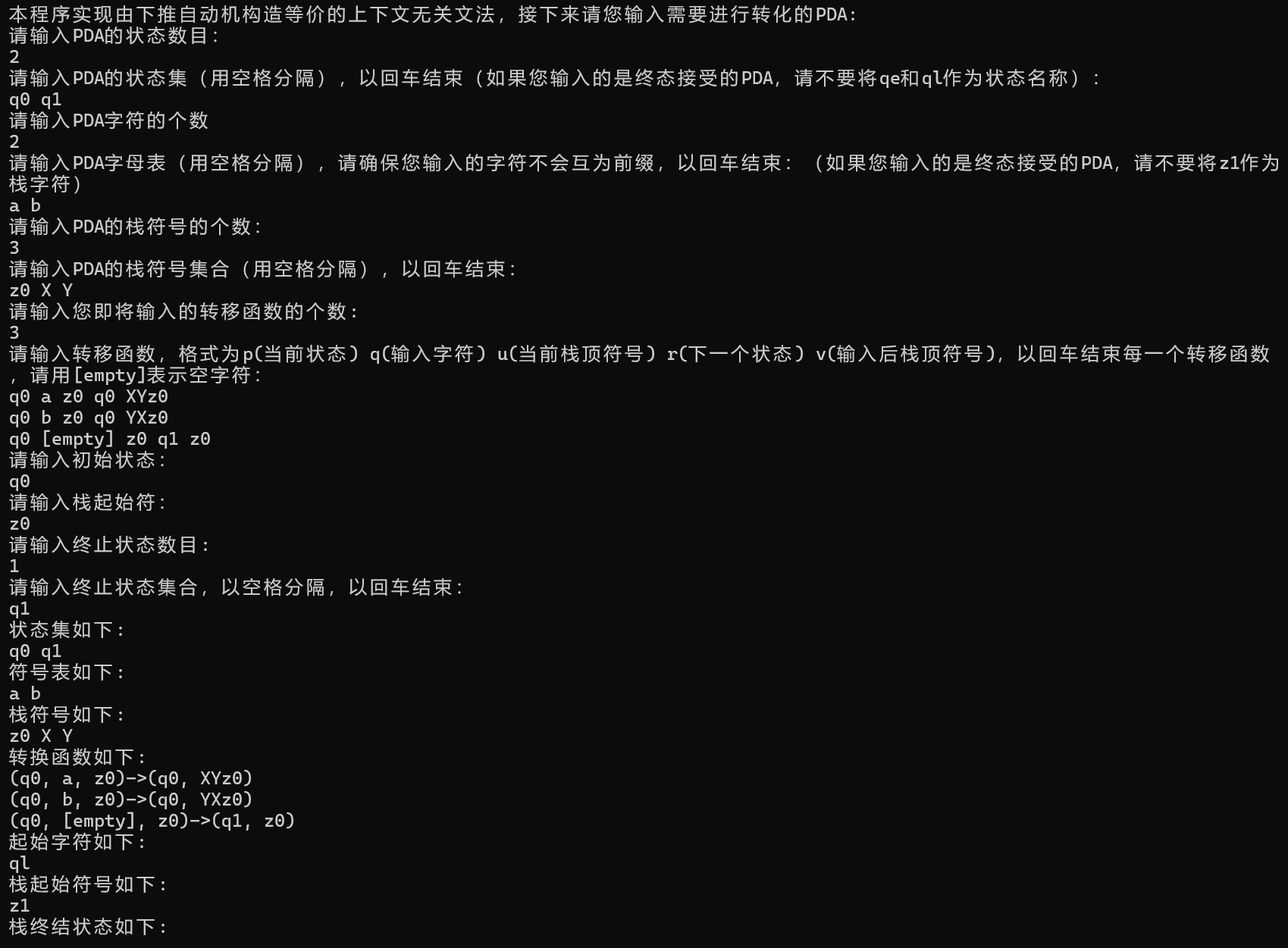


输出：

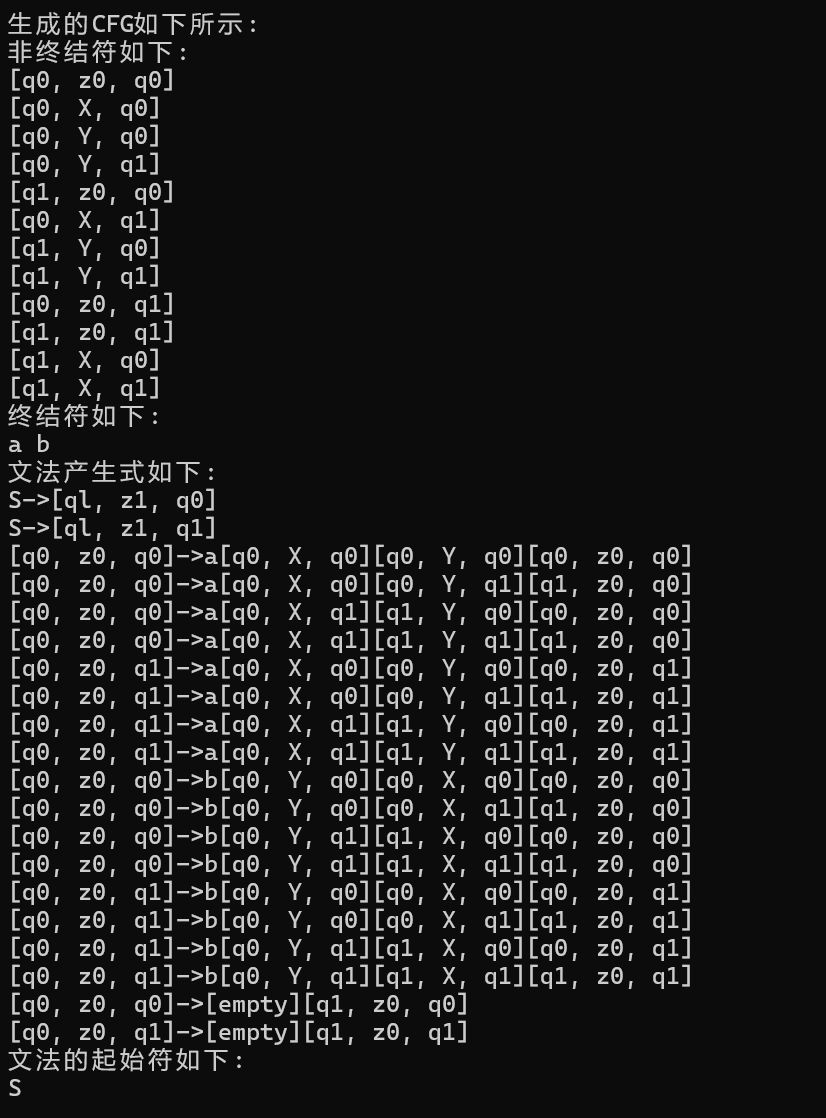


**用例三：（含有一次转换压两个字符入栈的PDA）**

输入：



输出：



## 实验总结

#### 5.1 本次实验学到的内容

（1）进一步熟悉了由PDA构造等价的CFG的过程和算法；

（2）细致了解了PDA的状态和CFG的非终结符、终结符的关系

（3）熟悉了c++ string、vector的用法

（4）熟悉了c++ "=="和"<<"运算符的重载

#### 5.2 程序进一步完善的地方

（1）因为循环遍历求所有状态序列的集合的算法复杂度过高，最多只考虑了一次压2个字符入栈的情况，可以考虑更多字符入栈的更简洁的算法；

（2）只考虑了对空栈接受字符串的PDA构造CFG，对于终态接受字符串的PDA，在做进一步完善时，可将其先转换成空栈接受字符串的PDA，在对其构造CFG （已完善，见3.3(5)）

#### 5.3 本次实验遇到的问题

问题：由于有的栈字符可能用两个字符表示，如z0，当栈顶字符有多个时，难以识别字符有几个、分别是哪个

解决方法：字符串匹配，构造vector<string> seperate，对每个栈字符，查找其是否在栈顶字符串中，若在，就根据出现的先后顺序存入seperate中，完成对栈顶的多个字符的分离。