编译原理课程设计报告

专题1：SysY语言的词法分析程序

一.实验目的

设计、编制并调试一个自定义语言SysY的词法分析程序，加深对词法分析原理的理解。

二.实验内容

1：自定义语言SYSY的的词法系统:

1)类型系统：支持int、char、void基本类型，分别用词法记号表示为关键字int、char和void。

2)常量：字符常量(用单引号括起来)、字符串常量（用双引号括起来）、八/十/六进制整数常量（0开头表示八进制，0x开头表示十六进制）。分别用词法记号表示为ch、str和num。

3)变量：与常量对应，使用标识符表示，词法记号表示为id。

4)表达式运算符：支持加减乘除、求余、取负、自增、自减算术运算，大于、大于等于、小于、小于等于、等于、不等于关系运算，与、或、非逻辑运算，表示为词法记号：‘+’，‘-’，‘*’，‘/’，‘%’，‘-’，‘++’，‘--’，‘>’，‘>=’，‘<’，‘<=’，‘==’，‘！=’，‘&&’，‘||’，‘！’。注意：取负运算和减法运算在词法分析器里是被看做是同一个词法记号。

5)语句：支持赋值语句、do-while、while、for循环语句，if-else、switch-case条件分之语句、函数调用、函数返回、跳转等语句。涉及的词法记号表示为赋值号‘=’，关键字do, while, for, if, else, switch, case, default, return ,break, continue。语句和函数体要求用大括号括起来，case和default后面需要跟冒号，因此需要包括各种分界符作为词法记号：‘{’，‘}’，‘;’，‘:’，‘(’，‘)’，‘,’。

2：词法分析程序的功能:

输入：所给文法的源程序字符串。

输出：二元组（编码，值）构成的序列。（能清楚表示识别出来的单词和它的类别）。

三．实验步骤

本实验总体设计思路：逐字符识别输入字符串。1遇到字符和数字时，将该字符放入str数组；2当遇到其他字符时,1先读入str数组内容：1.若首字母为字母，将其与关键字匹配，若匹配成功，则输出为关键字；若匹配失败，则输出为标识符id。2.若首字母为数字，则输出为数字常量；3.清空str数组等待下一次的输入。2判断若输入字符为运算符，获取它的下一位输入字符，若匹配双目运算符则输出双目运算符同时将定位变量i向后移一位，否则输出该字符为单目运算符。3判断若输入字符为分隔符，输出该字符为分隔符。4判断若输入字符为”，在未遇到匹配的”之前，将字符串中的字符都加入str数组中，匹配到”则输出为字符串常量，再将str清空，并将定位变量向后移n位。5若判断若输入字符为’，处理方式与字符串常量处理相同，输出为字符常量。

四．碰到问题及解决办法

1. 调试中发现，若输入源程序的最后一串为数字、标识符或者关键字时，无法输出。发现原因是将其输入到str数组后，由于没有遇到其他符号，便无法输出。于是在程序的最后加了一个模块用来输出以上这种情况。

2. 实验中对数字这一块的错误输入做了判别。例如八进制中输入0~7以外的字符就是错误的，十六进制中输入0~9、A~F(a~f)以外的字符就是错误的。

3. 验收时，发现数字0对应的是八进制常量，经过检查，发现问题。在数字识别时因为八进制是0开头且第二位不为x，于是错误的将八进制的条件设置为str[0]==0 && str[1]!=’x’，导致0输入时第一步直接进入八进制的判断当中并输出八进制数字常量。条件改进为str[0]==0 && str[1]!=’x’&& str.length()>1，即接解决了这个小问题，使之输出为十进制数字常量。

五. 测试分析

输入

```
#include<stdio.h>
int a;
int main()
{
    a = 10;
                b = 0x11ff;
                c = 027;
                // d = 20;
               /* e = 30;
              */

    if ( a><0 )
                {
        return 1a;
    }
    else{
        return 0;
    }
}
```

输出

```
( int , <标识符,1> )
( a , <标识符,2> )
( ;  ,  <分界符,4> )
( int , <标识符,1> )
( main , <标识符,3> )
( (  ,  <分界符,1> )
( )  ,  <分界符,2> )
( {  ,  <分界符,2> )
( a , <标识符,2> )
( = , <运算符,7> )
( 10 ,  数字 )
( ;  ,  <分界符,4> )
( b , <标识符,4> )
( = , <运算符,7> )
( 0x11ff  ,  数字 )
( c , <标识符,5> )
( = , <运算符,7> )
( 027 ,  数字 )
( ;  ,  <分界符,4> )
( / , <运算符,4> )
( / , <运算符,4> )
( d , <标识符,6> )
( = , <运算符,7> )
( 20 ,  数字 )
( ;  ,  <分界符,4> )
( if , <关键词3> )
( (  ,  <分界符,1> )
( a , <标识符,2> )
 ><为非法标识符
( )  ,  <分界符,2> )
( {  ,  <分界符,3> )
( return , <标识符,7> )
  1a;为非法输入
( }  ,  <分界符,3> )
( else , <关键词5> )
( {  ,  <分界符,3> )
( return , <标识符,7> )
( 0 ,  数字 )
( ;  ,  <分界符,4> )
( }  ,  <分界符,4> )
( }  ,  <分界符,4> )
```

六. 总结体会

通过本次实验，了解了词法分析器的作用并对其有了更深的认识。本实验应该算是比较简单，但还是在初次完成后遇到了各种问题，说明自己的认识还不够到位，在前期构思的时候不够严谨，以后会更加注意。

专题2：词法分析相关理论算法实现

一． 实验目的

1. 理解有限自动机的作用，进一步理解有限自动机理论
2. 设计有限自动机的表示方式，采用合理的数据结构表示自动机的五个组成部分
3. 以程序实现有限自动机确定化和最小化算法，提高算法的理解和实现能力

二． 实验内容

利用状态表和有限自动机的运行原理编写和设计程序，判断输入的自动机是DFA还是NFA，如果是NFA，利用子集法将其确定化，然后利用求同法或求异法将所得的DFA最小化。

三． 实验步骤

对于输入的自动机的每条边，记录下节点和转移状态，并输入终态节点。根据节点和转移状态判断是否为NFA，若为DFA则结束；若为NFA，记录下0状态的空闭包，并作为一个集合项，对该集合项求move(),并对其求空闭包，若为新集合项则新建之；循环直至没有新的集合项产生，并对所有集合状态进行重新命名，并记录包含终态节点的集合。对该NFA求最小化，重新命名状态后输出DNA。

四． 碰到问题及解决办法

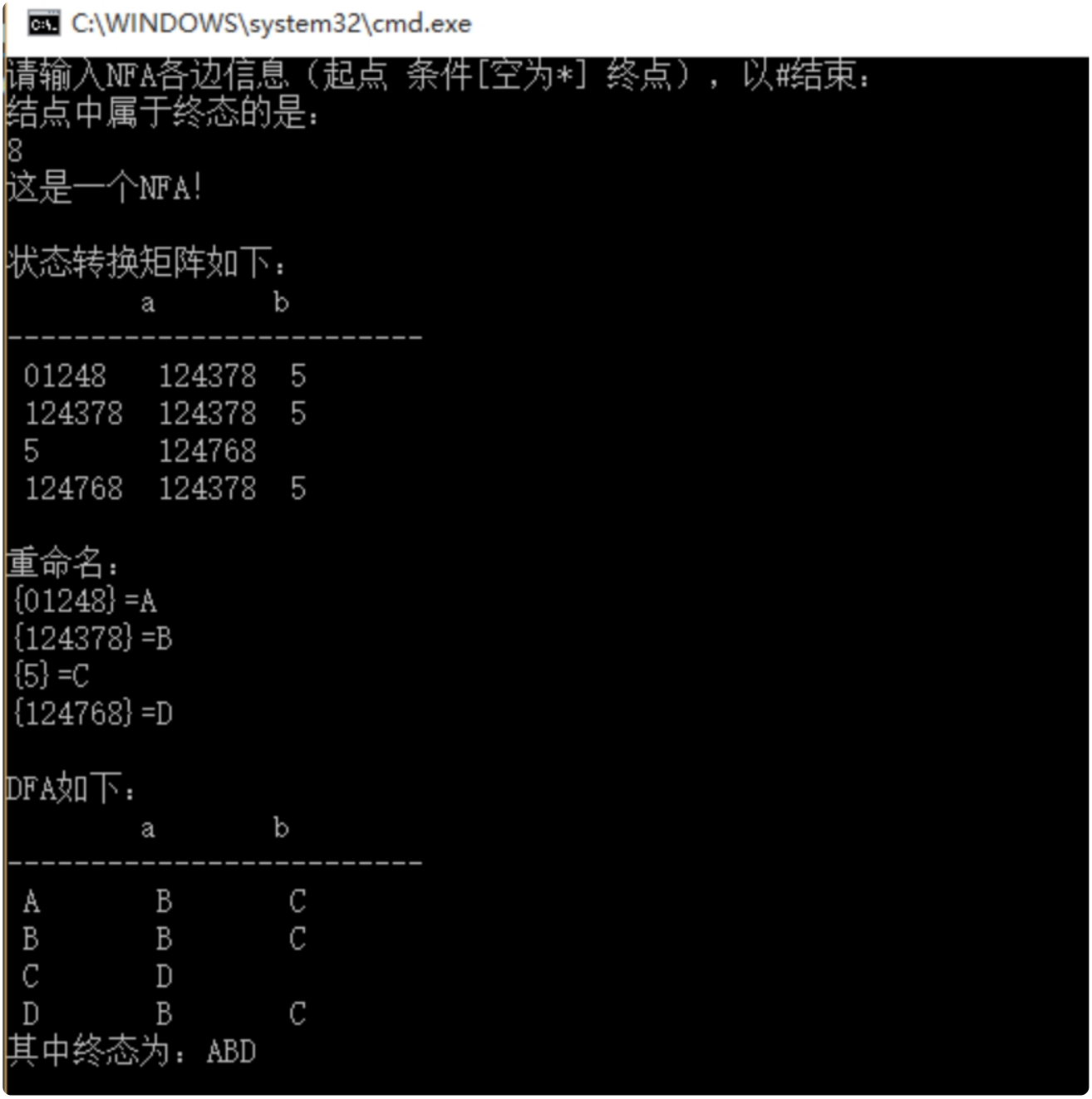
首先是对于每条边输入的数据结构存储问题，开始没有想到比较好的数据结构，询问验收通过的同学后找到了一个比较好的数据结构。

在判断NFA/DNA的过程中，要注意区别的两个条件（存在转移状态为空、一个状态接受到同一个转移条件能跳转到多个不同的状态）有一个符合就是NFA。

实验中，求move()时思路错误，误以为原集合接受到非空转移条件或者空转移条件都一起放入新的集合中，导致错误。正确思路是先接受到非空转移条件后将新集合中的状态进行求空闭包。

进行最小化时，在纸上实现比较容易，但是写代码实现时原来没有思路。参考pdf中的求异法完成模块。

五． 测试分析


C:\WINDOWS\system32\cmd.exe
请输入NFA各边信息（起点 条件[空为*] 终点），以#结束：
结点中属于终态的是：
8
这是一个NFA!
状态转换矩阵如下：
a b
01248 124378 5
124378 124378 5
5 124768
124768 124378 5
重命名：
{01248}=A
{124378}=B
{5}=C
{124768}=D
DFA如下：
a b
A B C
B B C
C D
D B C
其中终态为：ABD


集合划分：{C} {ABD}
重命名：
{C}=A
{ABD}=B
endnode:ABD
最小化DFA如下：
a b
A B
B B A
其中终态为：B
请按任意键继续. . .

六． 总结

该实验难度较大，在纸面上手动实现比较容易，虽然过程比较复杂。但是从纸面上转移到编程时，能力有限，遇到了很多困难，以后会增加这方面的锻炼。同时对NFA/DFA这块儿的内容有了更深的了解，也当是一次期末复习。

专题3：递归下降子程序的语法分析

一． 实验目的

掌握最基本的自顶向下分析方法，即递归下降子程序方法，理解其特点和适用范围（回溯，左递归等现象），锻炼递归调用程序的构造方法。

二． 实验内容

给定表达式文法G[E]：

E->E+T|T

T->T*F|F

F->(E)|i

根据该文法，编写递归下降分析子程序。

\1. 输入：任意的符号串(上述文法的终结符号”(”,”)”,”i”,”+”,”*”)

\2. 处理：调用递归下降分析程序

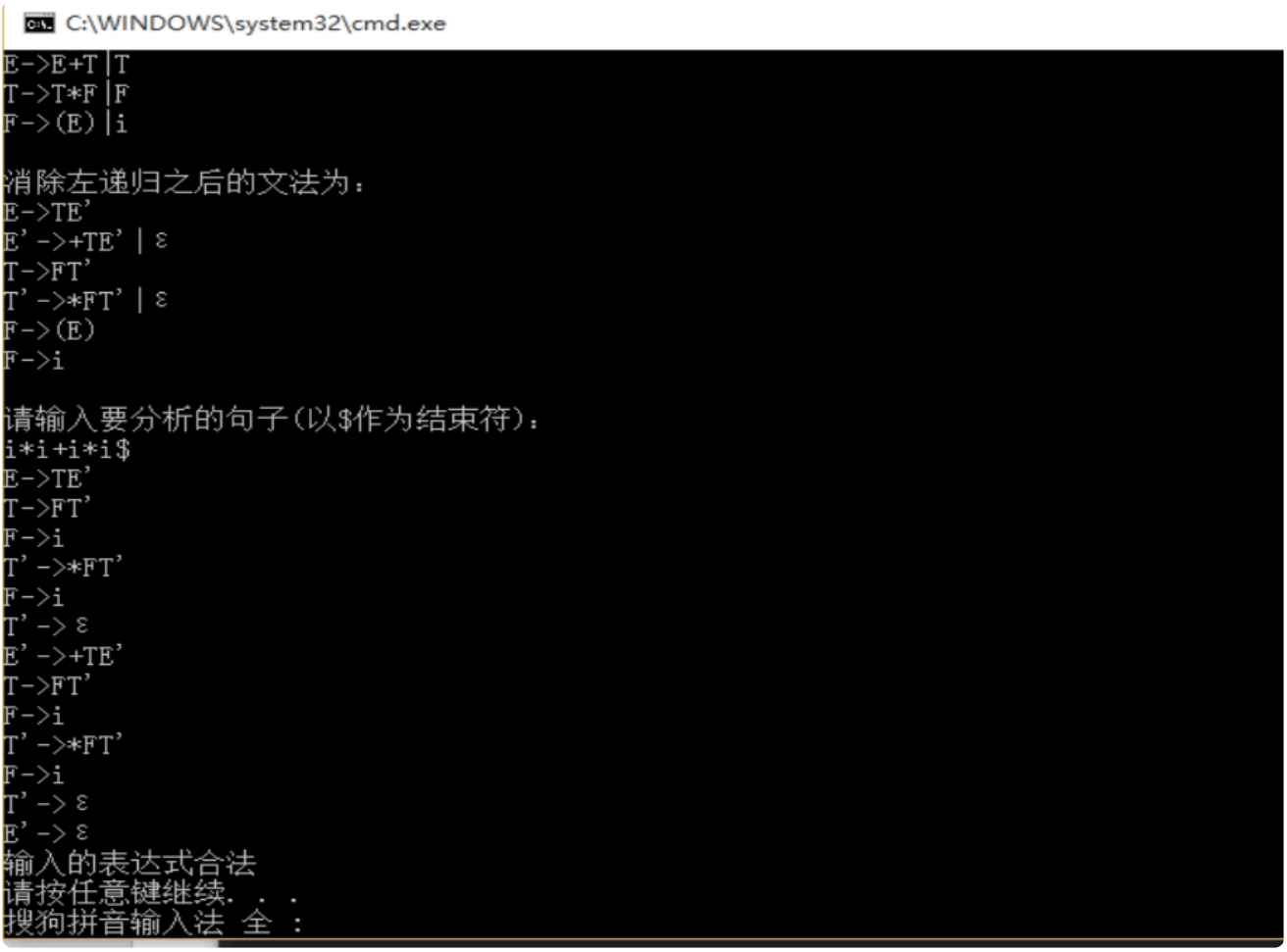
\3. 输出：判断输入串是否为合法表达式

三． 实验步骤

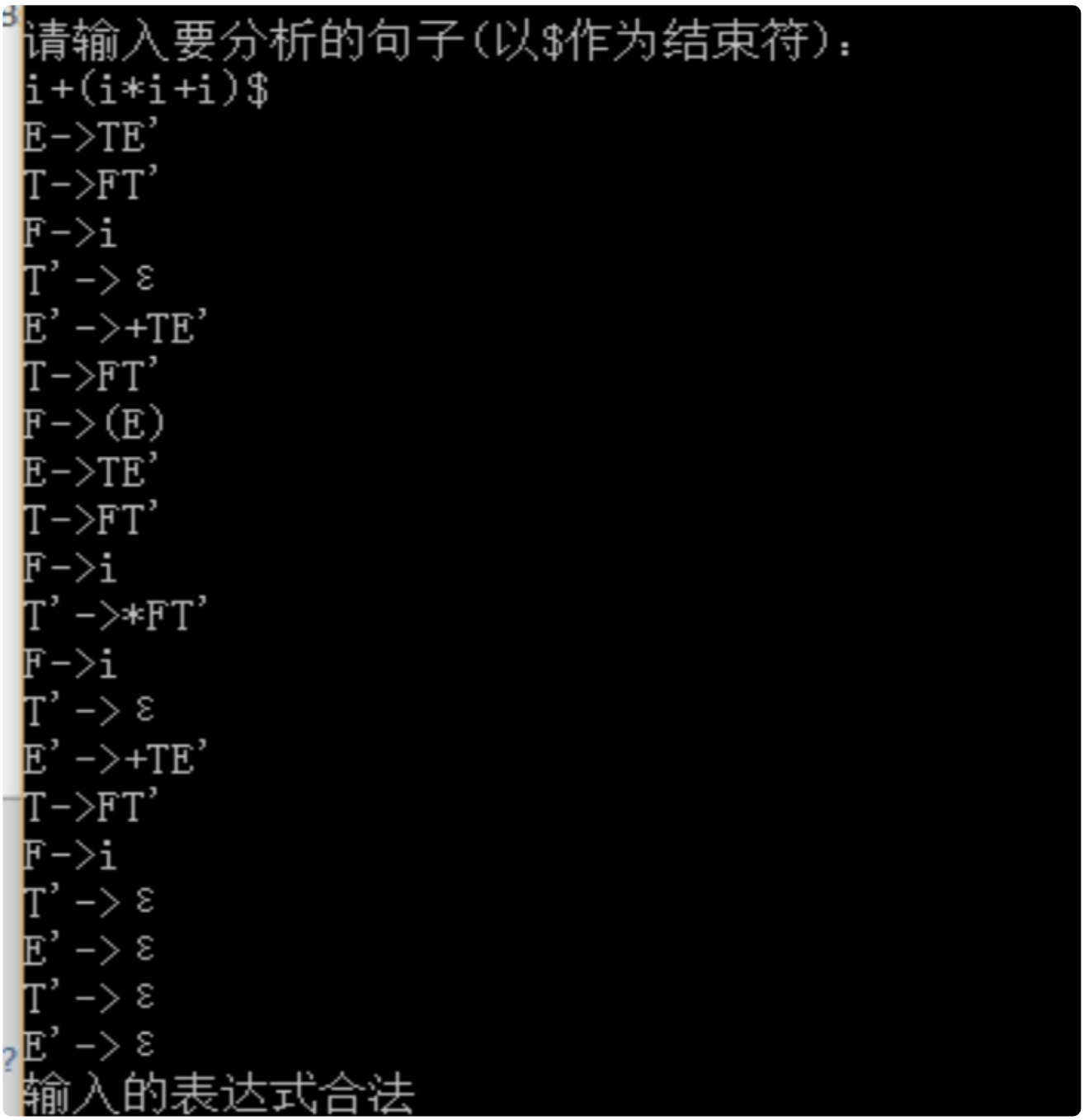
本实验总体设计思路：在纸上事先将上述文法消除左递归，再根据该文法编写左递归下降程序（只针对该文法）。例如E—>E+T|T消除左递归后为E->TA，A->+TA|#。则构造函数E(),依次调用T(),A()。构造函数A(),在符合+先输入的条件下，依次调用T(),A()。

四．测试分析

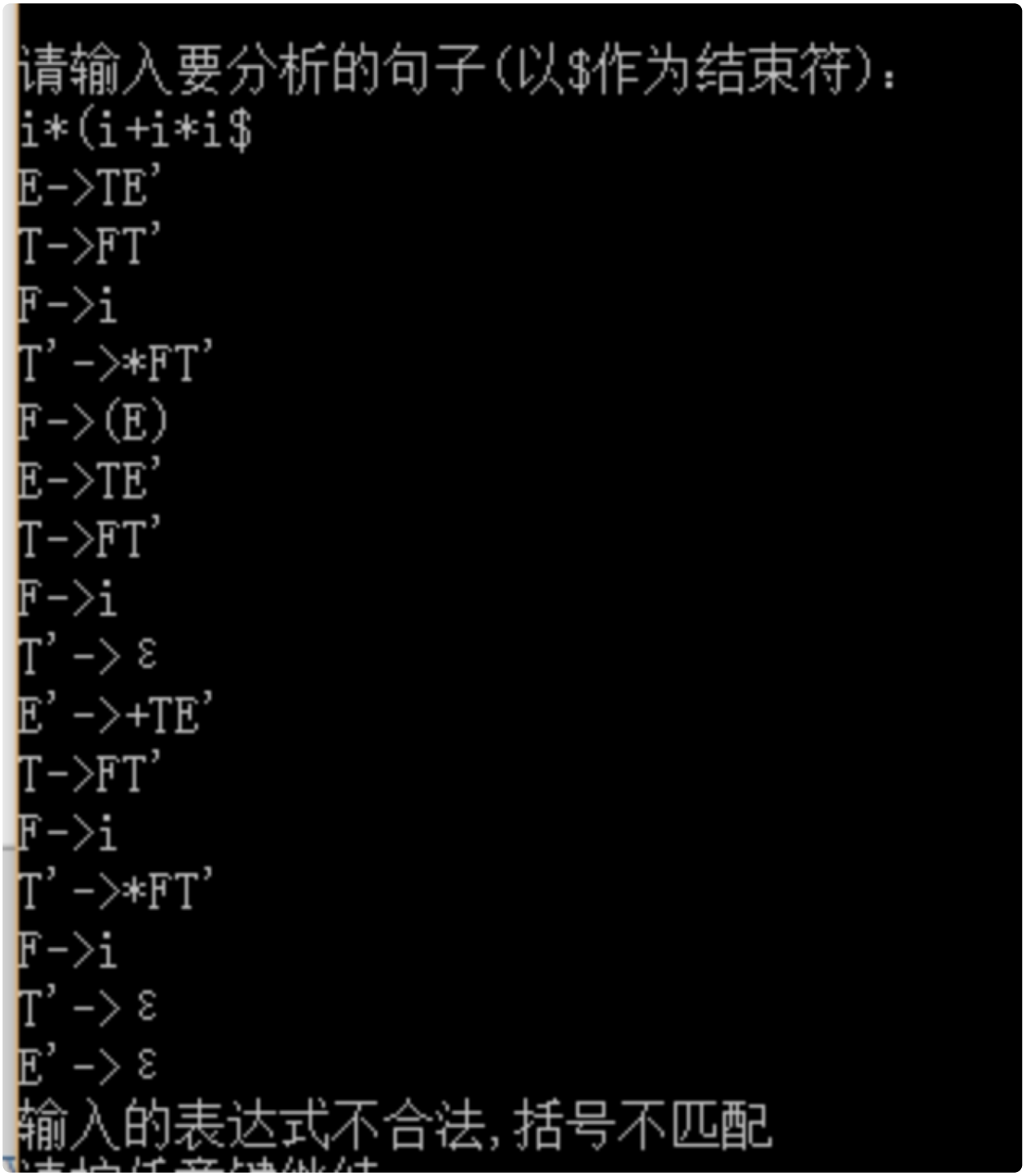
输入正确语句i*i+i*i$:



输入正确语句：i+(i*i+i)$


请输入要分析的句子(以$作为结束符):
i+(i*i+i)$
E->TE'
T->FT'
F->i
T'-> ε
E'->+TE'
T->FT'
F->(E)
E->TE'
T->FT'
F->i
T'->*FT'
F->i
T'-> ε
E'->+TE'
T->FT'
F->i
T'-> ε
E'-> ε
T'-> ε
E'-> ε
输入的表达式合法

输入错误语句：i*(i+ii$



六.总结体会

本实验虽然看似简单，但是我做的代码只针对要求中的文法，没有普遍适用性。而且左递归没有消除，但是在实验四中有左递归消除的模块。若要针对所有文法试用，则需要先消除左递归，并记录终结符号以及非终结符号，识别之并做递归分析，难度增大很多。

专题4：LL（1）语法器的分析与实现

一． 实验目的

1. 了解 LL(1)语法分析是如何根据语法规则逐一分析词法分析所得到的单词，检查语法错误，即掌握语法分析过程。
2. 掌握 LL(1)语法分析器的设计与调试。

二．实验内容

文法G[E]：E→TE’，E’→+TE’|ε，T→FT’，T’→*FT’|ε，F→(E) | i

针对上述文法，编写一个 LL(1)语法分析程序：

1. 输入：诸如 i+i*i 的字符串，以$结束。
2. 处理：基于分析表进行 LL(1)语法分析，判断其是否符合文法。
3. 输出：串是否合法。

三．实验步骤

本实验设计思路：先判断输入的文法是否存在左递归，若有则消除之，并将文法放入数组GG[][]中；若不存在左递归，则将原文法放入数组GG[][]中。对GG[][]中的文法依次求解FIRST集、FOLLOW集、SELECT集，再根据SELECT集构造对应的预测分析表。对于输入的串，根据预测分析表来检查输入的文法是否正确。求解各个集合的结果以及预测分析表及匹配过程在界面中给出。

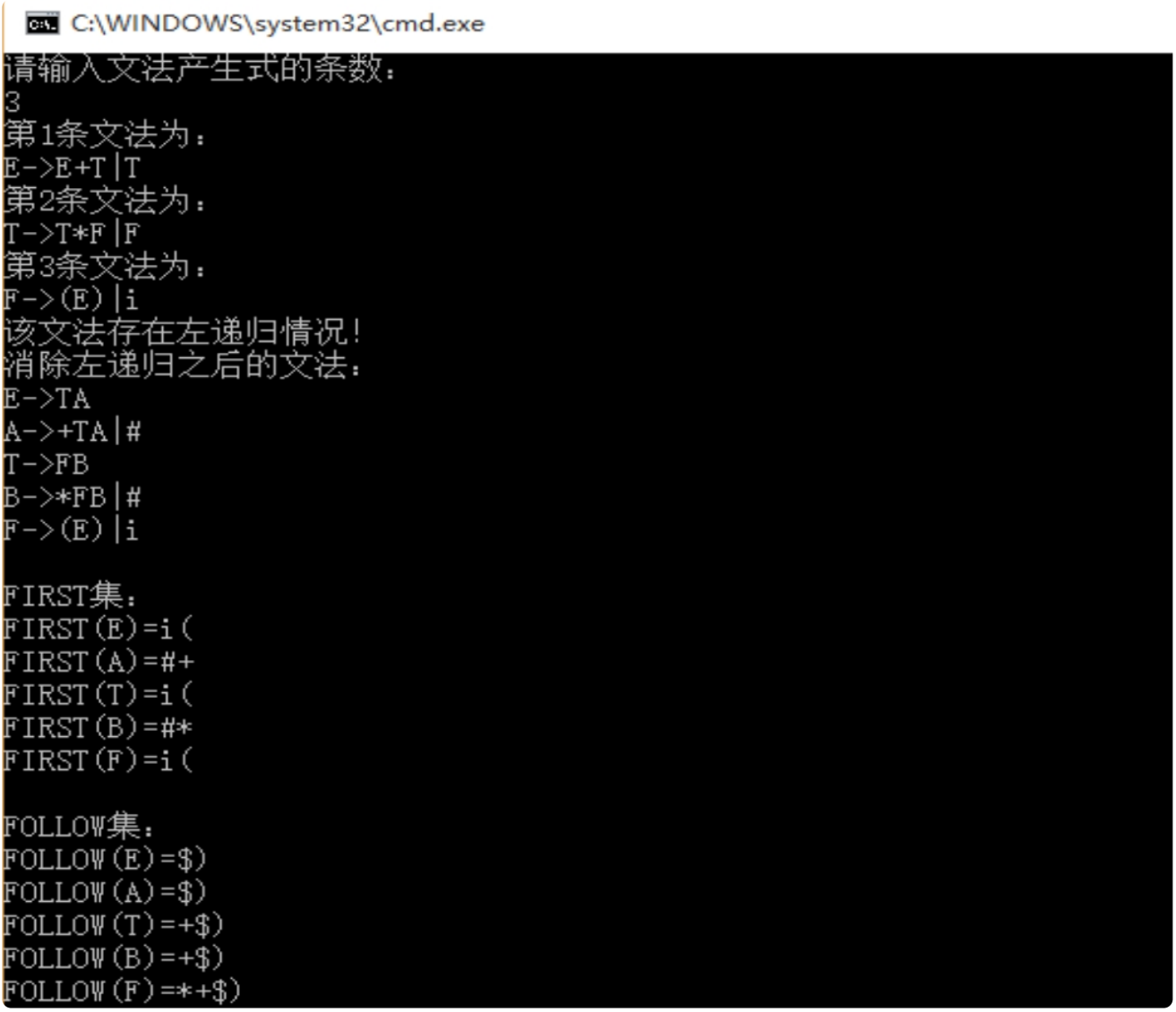
四．碰到问题及解决办法

1求解FIRST集时，刚开始发现无法设置参数检验一次文法执行完后是否每个FIRST集有变化，于是索性直接循环文法执行10次(假设10次以内FIRST集不再有变化)。在验收时询问老师发现可以在某个FIRST集添加元素时将flag置为1表示有变化，一次文法循环完后，若flag值为1则循环该文法直至某次flag值为0。FOLLOW集也可以采用这种方法，在试验验收后重新改修改程序完成了用flag记录的方法。

2平时在作业中FIRST集和FOLLOW集很好求，但是在编程时自己还是想的不够有条理，借助了书上和老师所给pdf中的算法思想完成。

3在最后根据预测分析表检验输入串的过程中，因为懒，用了STL中的stack，但是在每次匹配之后输出分析栈剩下来的内容时，竟然，没有直接输出stack中全部元素的函数!(在Java中可以直接输出stack中的元素而且不需要调用函数)…还得写一个没有难度的输出栈所有函数也是够了…

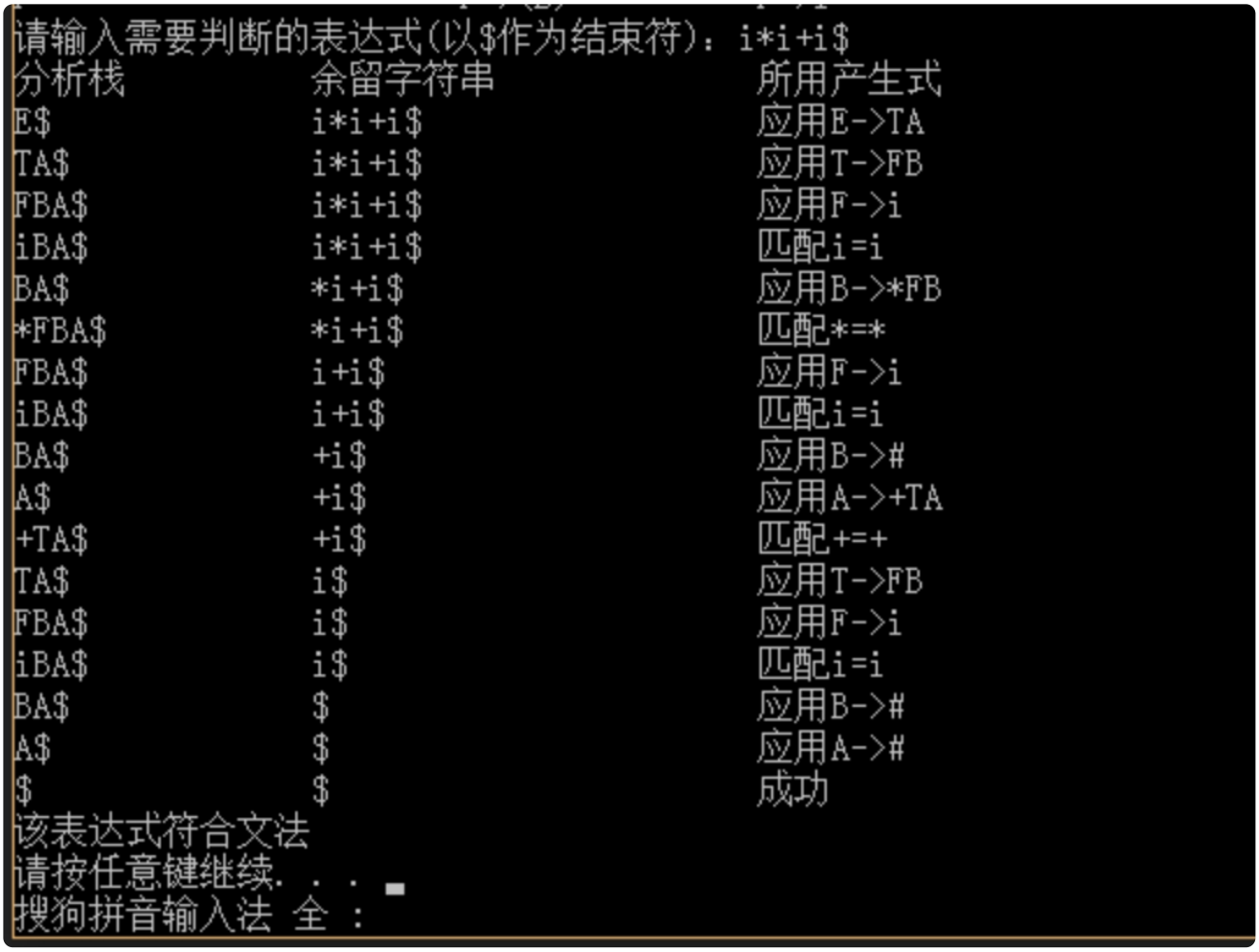
五．测试分析



SELECT集：
SELECT(E->TA)=i(
SELECT(A->+TA)=+
SELECT(A->#)=$)
SELECT(T->FB)=i(
SELECT(B->*FB)=*
SELECT(B->#)=+$)
SELECT(F->(E))=(
SELECT(F->i)=i

预测分析表：

	+	*	(	)	i	$
E			E->TA		E->TA	
A	A->+TA			A->#		A->#
T			T->FB		T->FB	
B	B->#	B->*FB		B->#		B->#
F			F->(E)		F->i	



六．总结体会

虽然实验要求针对该文法，但是其实是针对所有文法都要成立，因此难度有所增大，因为自己的编程能力有限写了好久才写出来，而且没有分成模块(完成后有做过分开成模块的工作，但是失败了)，代码的可读性不强，以后会注意提升自己。通过这次实验，对于FIRST集、FOLLOW集、SELECT集和LL(1)文法有了更深刻的理解和系统的求解过程了解。