|  |
| --- |
| **成绩** |
|  |



中国农业大学

编译原理实验报告

（2019-2020学年春季学期）

课程名称： 编译原理实验

任课教师： 王耀君

班　　级： 计算 172

学　　号： 2017304010417

姓　　名： 郭际虎

目录

[综合实验 PL0扩充 2](#_Toc43069142)

[**一、实验目的** 2](#_Toc43069143)

[**二、实验环境** 2](#_Toc43069144)

[**三、实验内容** 2](#_Toc43069145)

[**四、实验要求** 2](#_Toc43069146)

[**五、具体实现** 3](#_Toc43069147)

[**六、扩充功能说明** 4](#_Toc43069148)

[运行结果总览： 4](#_Toc43069149)

[扩充部分语法图以及BNF 4](#_Toc43069150)

[1、新增数据类型：dvar、array、tuple 8](#_Toc43069151)

[2、新增格式化输入输出 11](#_Toc43069152)

[3、新增for循环语句 14](#_Toc43069153)

[4、各类型运算部分 17](#_Toc43069154)

[5、错误处理部分 20](#_Toc43069155)

[6、其他修改部分 21](#_Toc43069156)

[7、配置部分 22](#_Toc43069157)

[**六、心得与思考** 26](#_Toc43069158)

综合实验 PL0扩充

**一、实验目的**

本综合实验要求学生根据已经掌握的编译技术，读懂读通世界著名科学家 先生编写的 “PL/0 编译程序”，并根据要求修改编译源程序，在符合 PL/0 语言基本词法、语法规则的前提下，对PL/0 语言的功能进行扩充。同时考察 学生设计和撰写实验报告和汇报PPT的能力。

**二、实验环境**

、、。

**三、实验内容**

1. 本综合实验要求在解读PL/0 编译程序的基础上，学习 PL/0 语言语法规则、程 序结构、词法分析、语法分析和目标代码结构、代码生成、语法错误处理和目标代码在执行时的存储分配。进一步理解编译程序中的各个组成模块的功能，并且能够对编译的过程有个整体的把握。

2. 根据实验要求对PL/0的语言功能扩展，对PL/0 语言编译程序修改，完成实验要求的基本功能，并对实验扩展功能模块的编译；可以增加除实验要求外的其它功能，鼓励创新。

3. 修改后的PL/0 语言编译程序，可以编译通过并运行。对于给定的测试用例可以得到正确的目标程序和运行结果。

4. 实验报告撰写要形式规范，内容组织详实且富有逻辑性；汇报PPT要突出工作重点，内容组织要具有逻辑性，图片和动画等素材编排美观，可体现整个实验工作的重点和亮点。

**四、实验要求**

1. 使用 或其它工具画出PL/0编译程序主程序、各个函数的程序 流程图，并简要说明功能。（6分）

2. 基本功能扩展（24分）

（1）I/O的功能扩展：增加格式化输入、格式化输出。（6分）

（2）数据结构的扩展：增加浮点数、向量、矩阵等。 （6分）

（3）计算上的功能扩展：增加对浮点数运算、向量运算、矩阵运算的支持等。 （6分）

（4）控制逻辑上的扩展： 增加循环，适用于 和 循环的语 句、语句、 语句功能等。（6分）

3. 高级功能扩展（10分）

以下条目至少完成一项：

（1）出错处理：增加以上新增基本功能的出错处理。

（2）内存管理：增加垃圾回收机制、最大内存可配置功能。

（3）代码优化：扩充代码方面优化功能，使用局部优化、循环优化、全局优化的思想。

（4）设计：完成基于 或 等编辑工具的语言集成开发环境配置，实现代码折叠、代码缩进、自动补全、语法高亮、显示行号等功能。

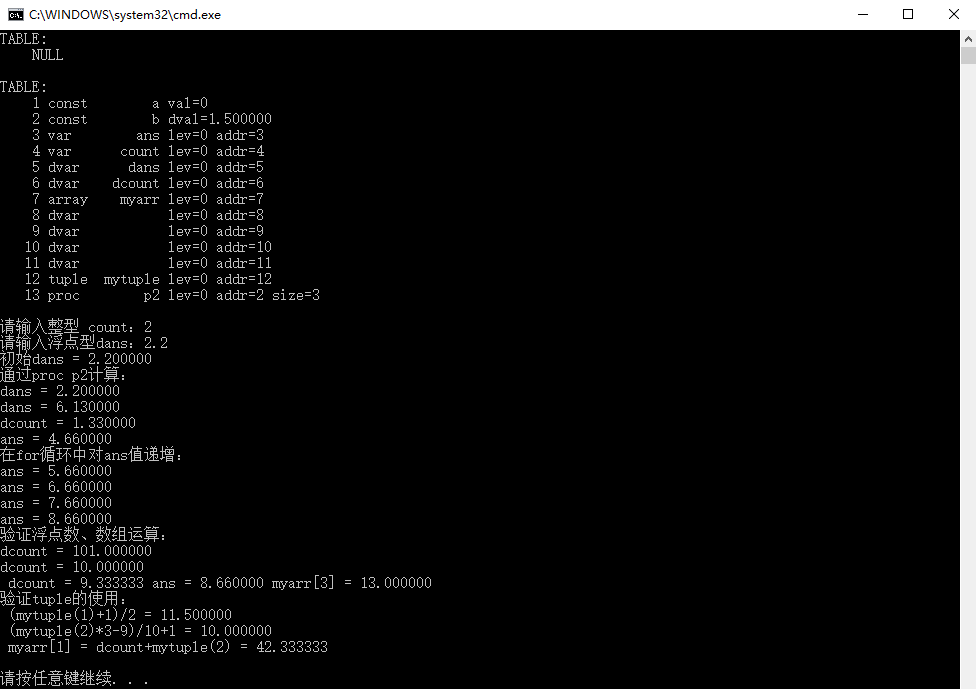
4. 报告及PPT撰写（10分）

（1）实验报告：注意格式、逻辑、素材、引文等。（5分）

（2）报告PPT：注意格式、逻辑、素材等。（5分）

1. **具体实现**
2. 格式化输入。
3. 格式化输出。
4. 增加浮点数。
5. 增加一维数组（向量）。
6. 增加元组（需要在创建时指定值）。
7. 支持整型、浮点数、一维数组、元组之间的计算。
8. 实现for循环语句。
9. 上述功能对应的错误处理。
10. 基于的部分功能。
11. **扩充功能说明**

运行结果总览：



**验证元组运算**

**验证浮点数、数组运算**

**利用for循环对于ans的值进行修改**

**格式化输入效果**

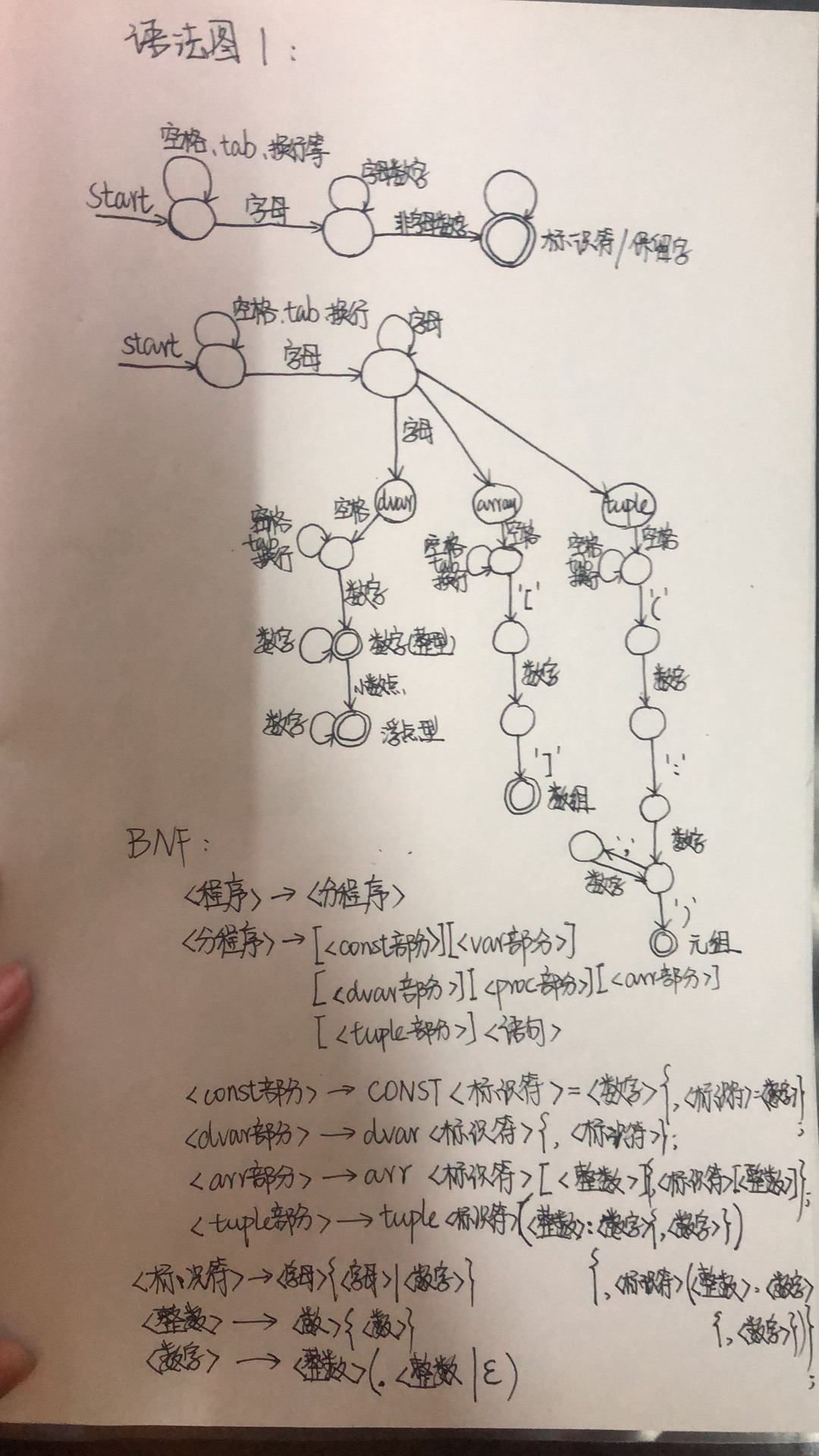
**函数调用计算浮点数和整型数**

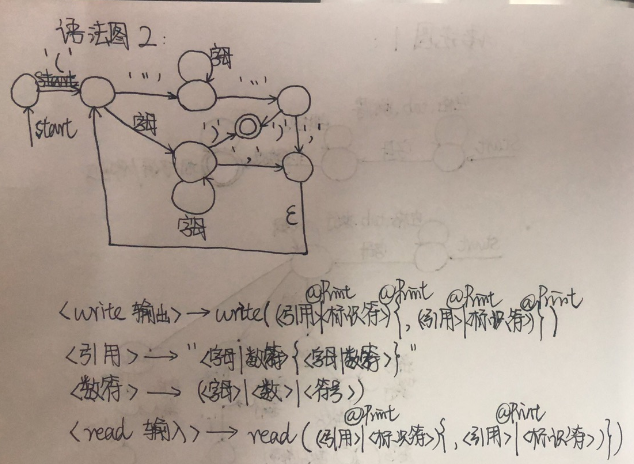
**实现浮点数dvar、数组array、元组tuple**

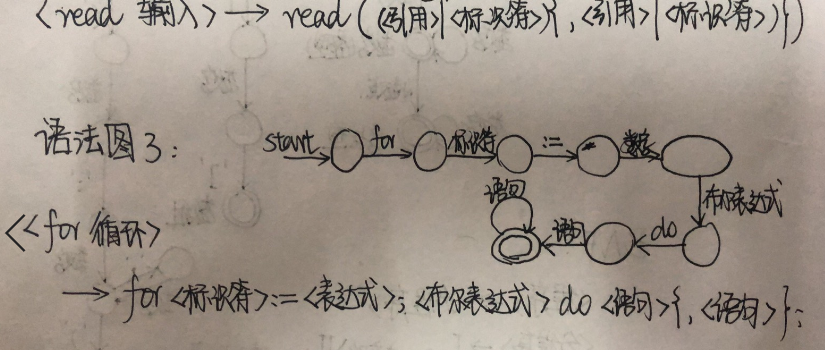
**测试PL0程序**

扩充部分语法图以及BNF

**如下图所示：**

****

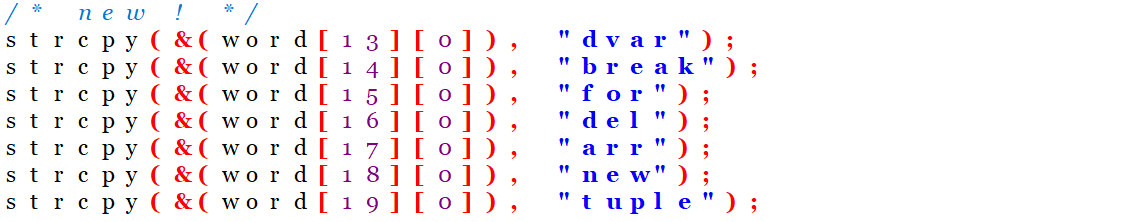


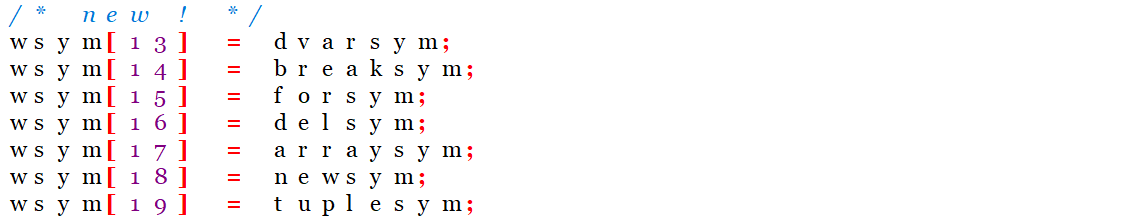


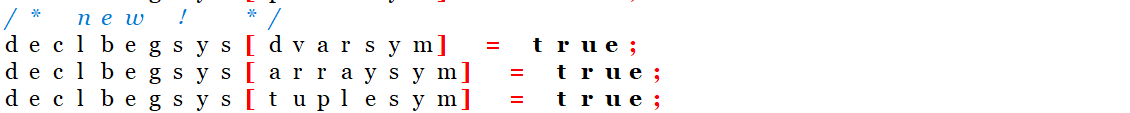
新增符号判断：



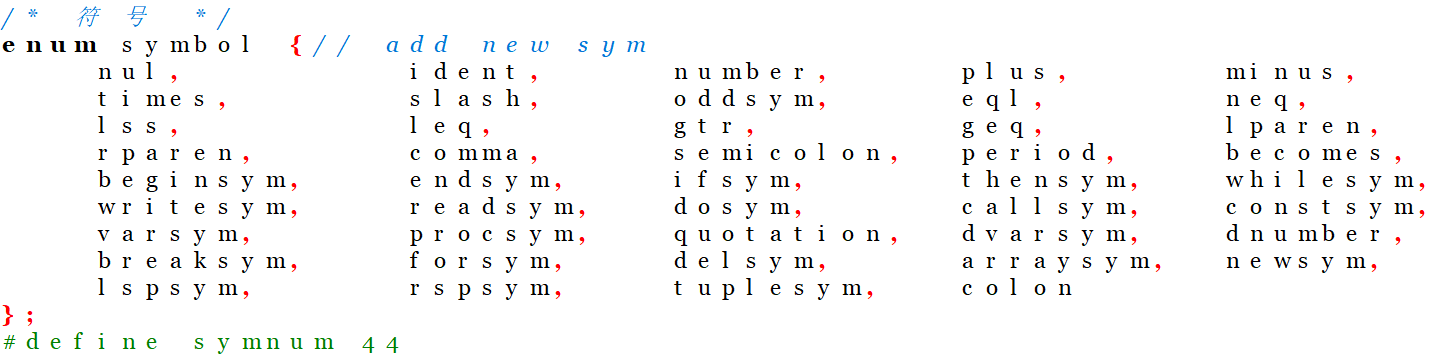
新增关键字：







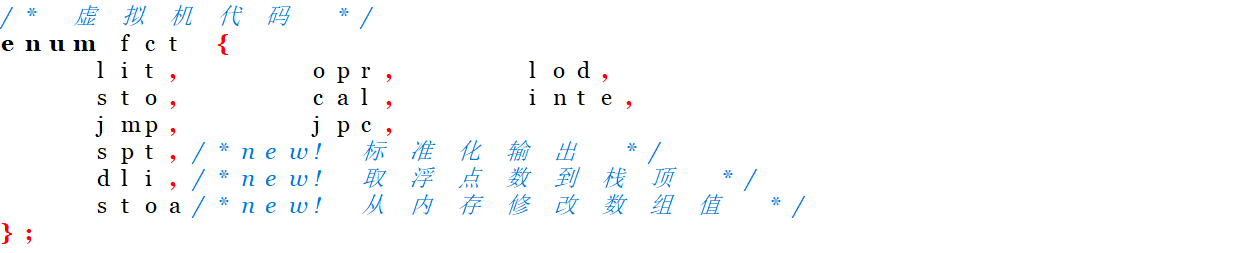
完整符号表：



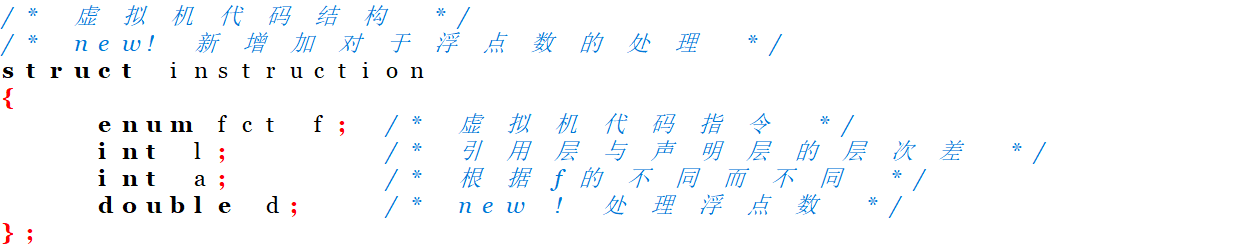
名字表扩充：



虚拟机指令扩充：



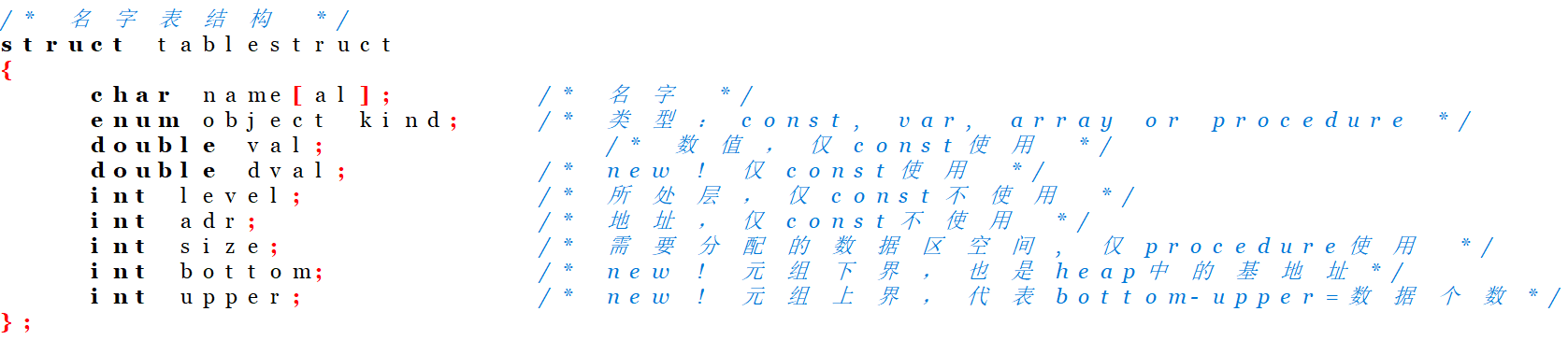
虚拟机指令结构变动：增加double类型数据用于处理浮点数。



新增全局变量：



修改名字表结构体：



新增堆，用于存储元组数据：

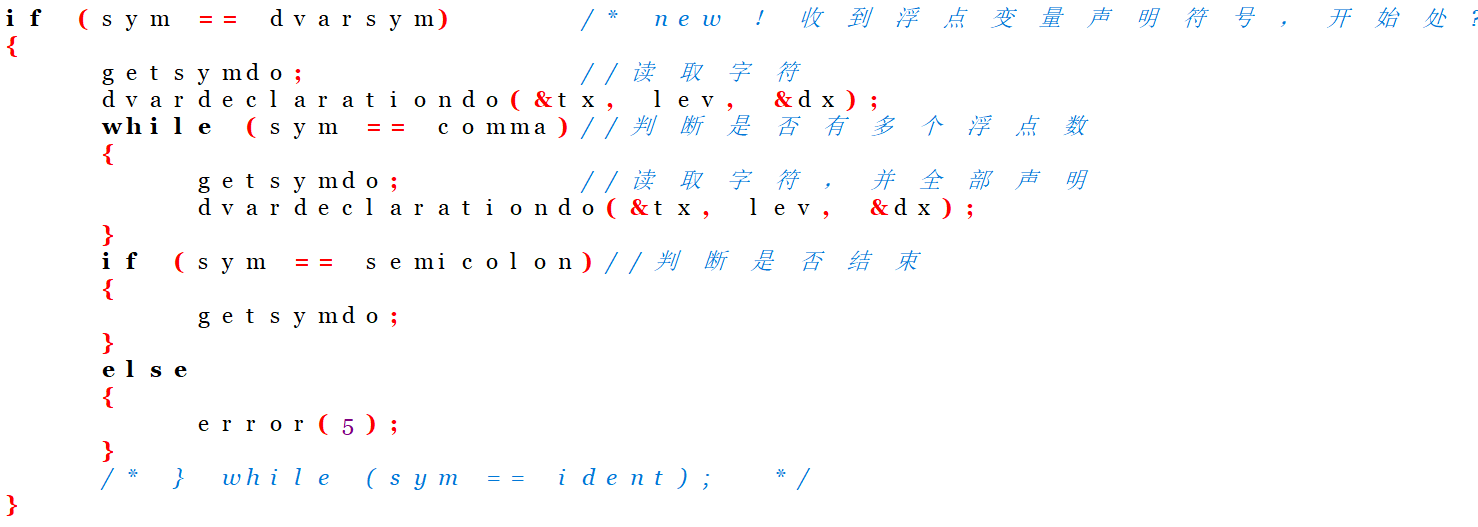


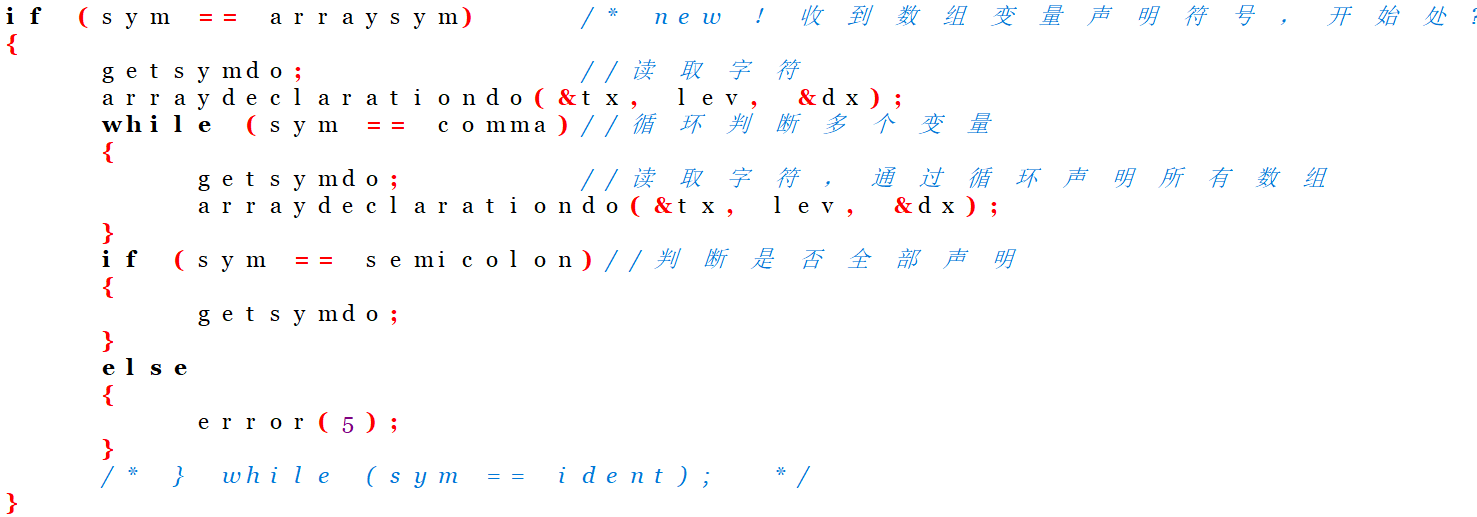
新增格式化输出缓冲区：

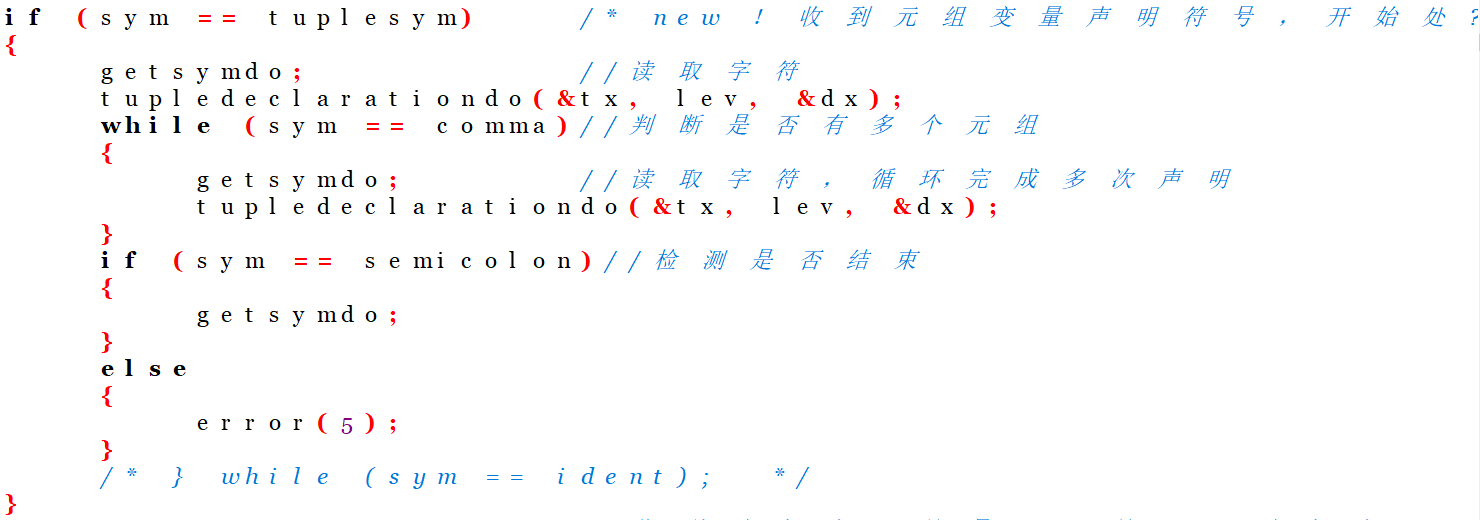


1、新增数据类型：dvar、array、tuple

对于新增数据类型首先在声明阶段需要进行扩充，包括对于dvar的声明函数、array声明函数、tuple声明函数：

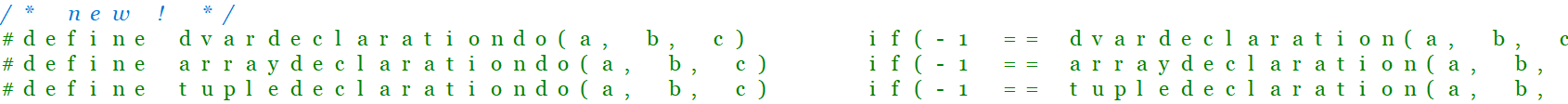




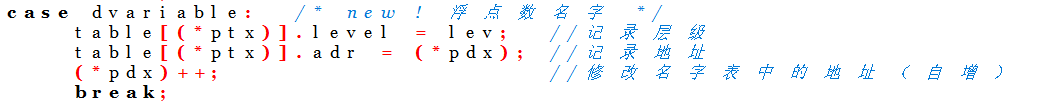


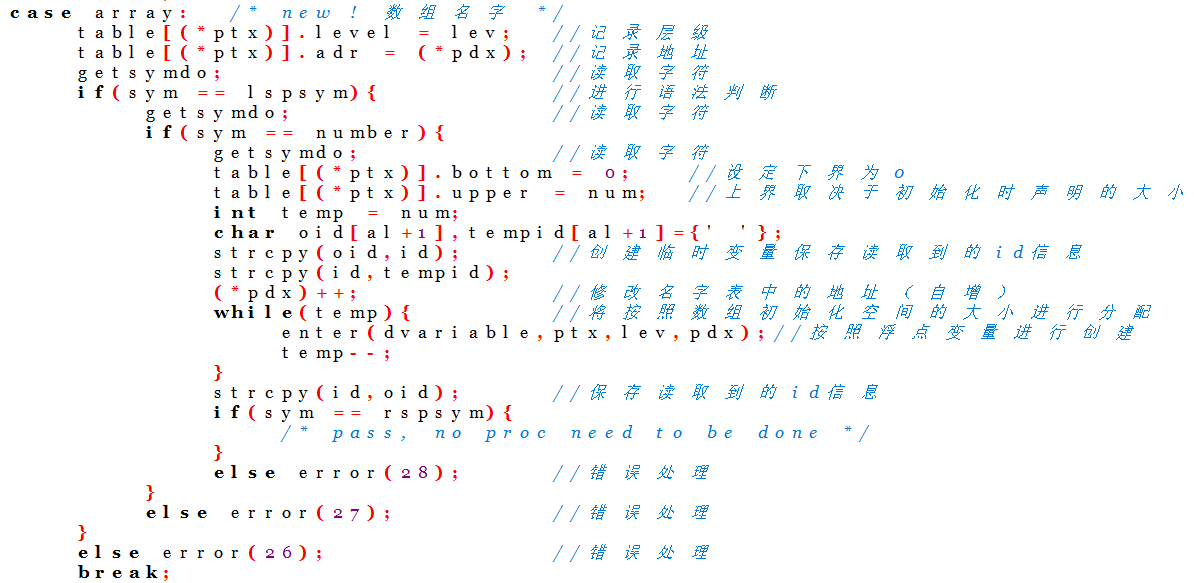
**Block()函数中添加三个扩充的声明函数捕捉代码**

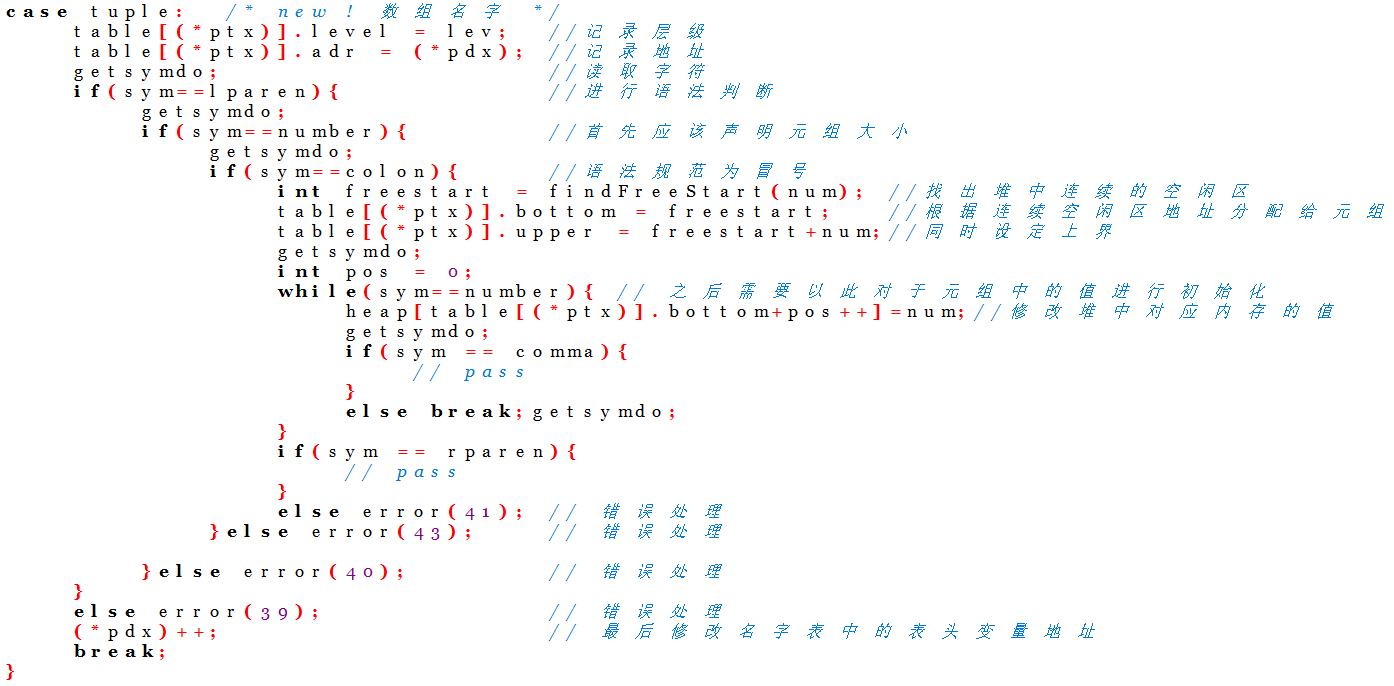
调用对应的声明函数：在声明函数中通过enter将扩充变量添加到名字表中。



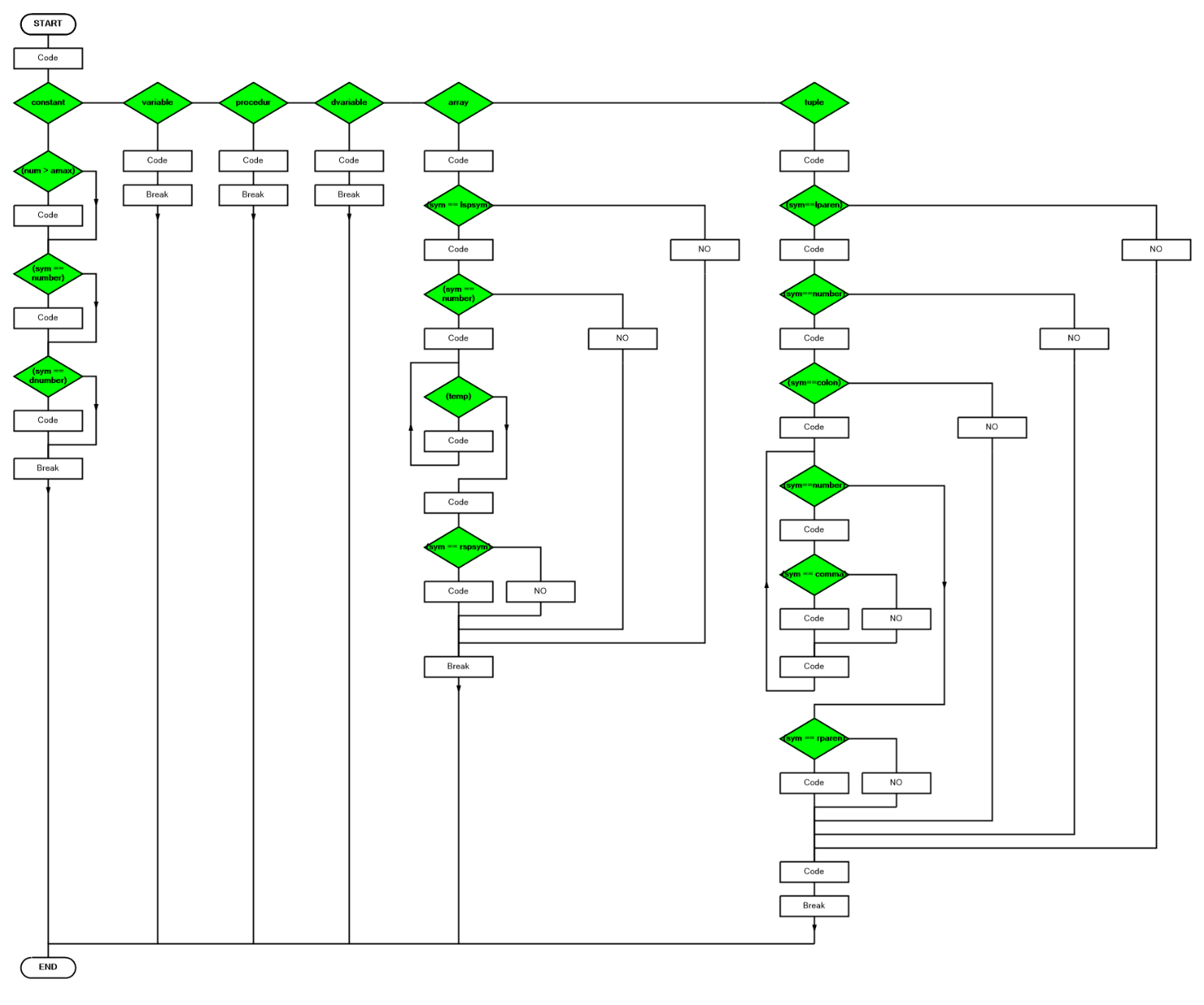
**此时需要修改enter()增加case情况判断：**





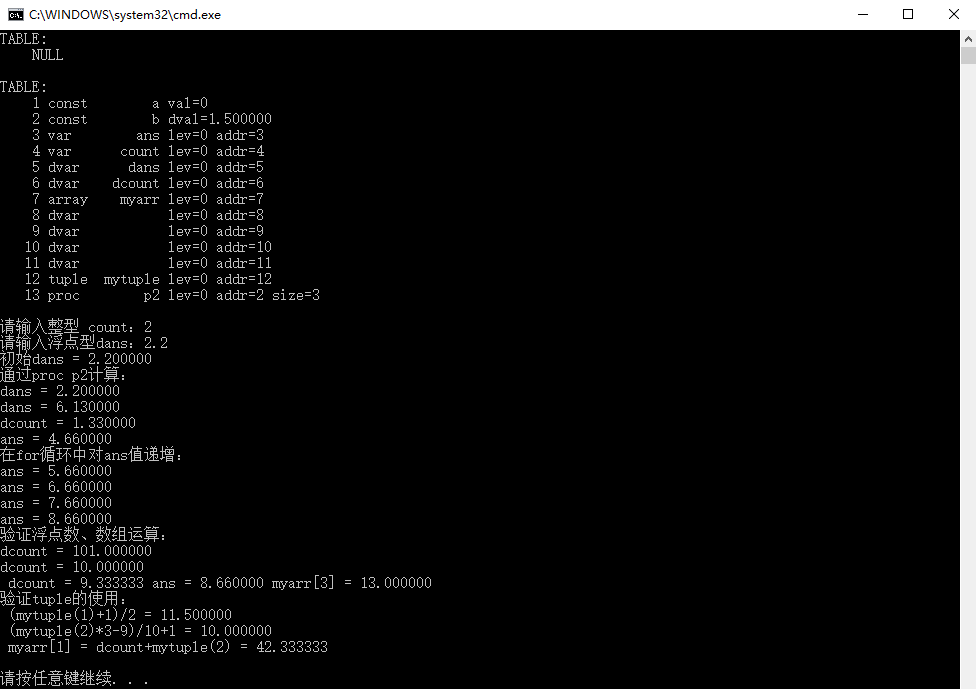


**新增enter()函数case情况判断**

****

**新增enter()函数流程图**

之后新增变量就完成了名字表的添加操作。关于del和new功能，一开始设想的是对于元组可以进行del和new操作，由于元组实际上是由新开辟的内存空间存储，所以可以借此利用最佳匹配空间分配方法进行空间优化，但是意义不大，可以直接调用指令获取常量值，del和new操作只是对于元组的名字在名字表中进行删除和添加操作，同时根据数组判断空闲情况、分配情况。



**实现浮点数dvar、数组array、元组tuple**

2、新增格式化输入输出

格式化输入实际上是格式化输出的功能以及原有的格式化输入操作的结合。所以先介绍格式化输出功能：



格式化输出主要涉及到字符串的调整，首先在可以多个变量同时输出的基础上加入字符串缓冲区：

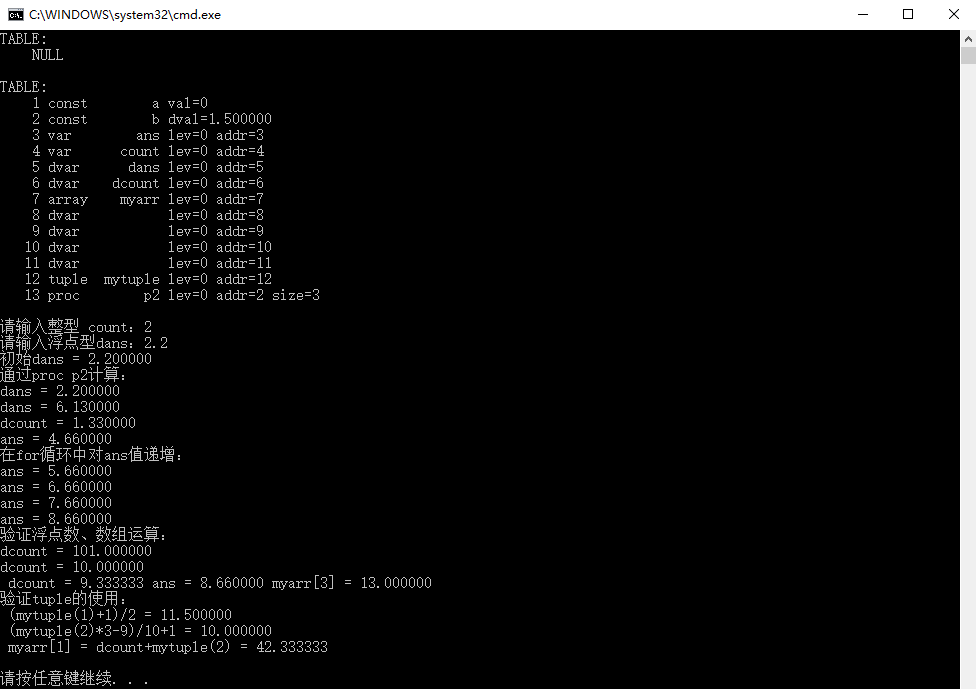


限制大小来保证内存不浪费，对于同样的字符串，比如循环语句中出现的输出操作应该可以识别。对原本的write()函数进行修改添加条件判断，判断即将输出的母标是变量还是引用（字符串）。若是引用则利用缓冲区进行缓存，在interpret()的时候调用指令进行输出，若是变量ident就从名字表中寻找对应的变量之后输出。最后两种分支都需要默认输出换行，对应的指令是（OPR，0，15，0）。

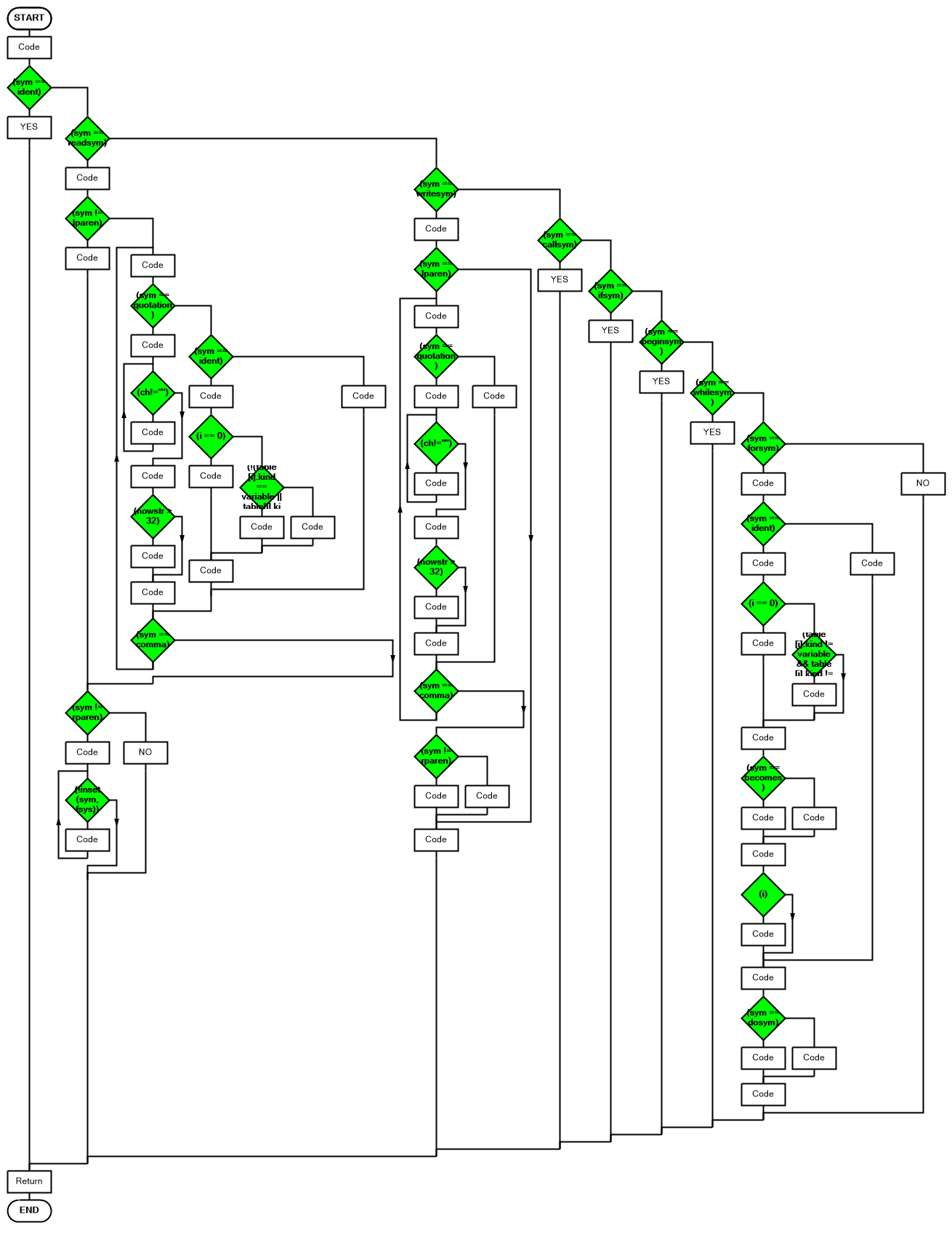
对于格式化输入语句则是融合了格式化输出的功能：



相当于python中的input("Please input a number")。



**格式化输入和格式化输出**

****

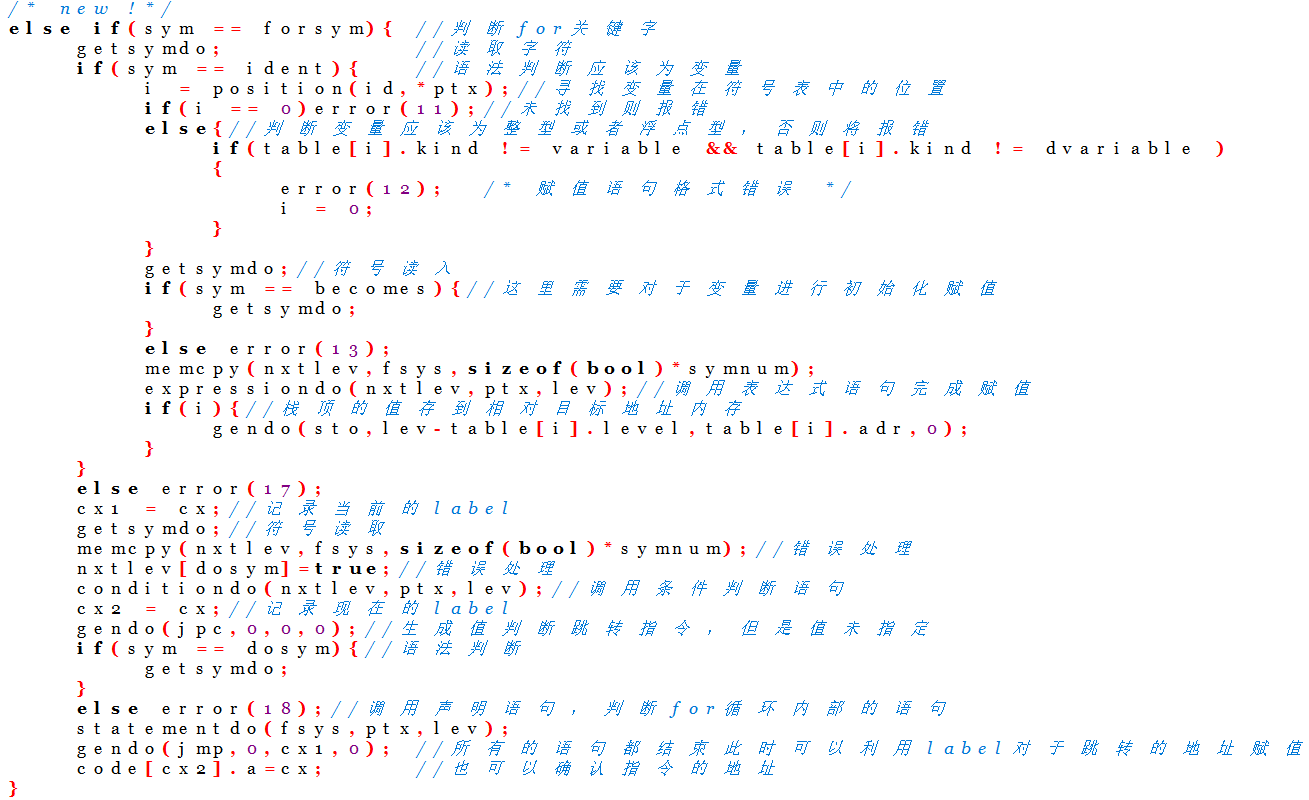
WRITE语句

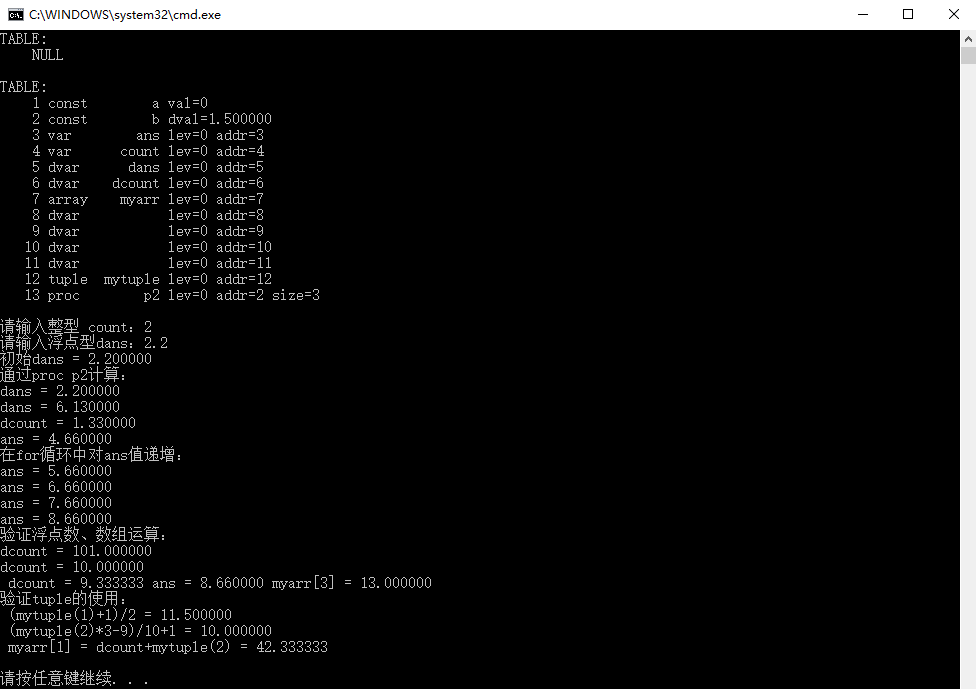
READ语句

**write()、read()函数流程图**

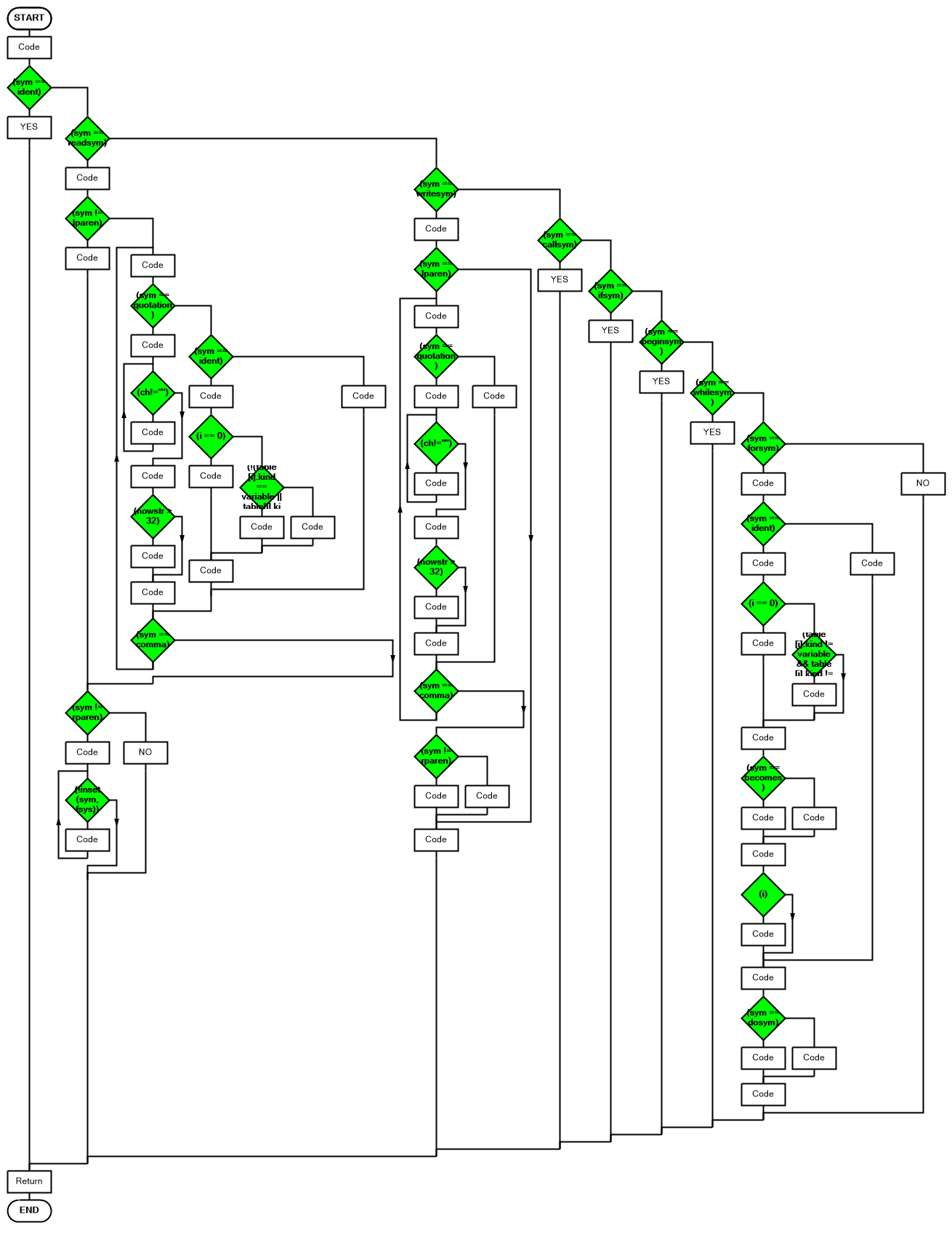
3、新增for循环语句

实际操作中for循环语句比较类似于while循环语句，在判断流程中需要利用到if条件跳转的技巧。区别在于，for循环中多了一个初始化变量的操作，该项操作对应到while中为布尔表达式的确认。在这个基础上同样需要使用临时变量存储指令的位置，当循环体内部的语句全部分析完毕后才可确定指令跳转的具体地址。





**利用for循环对于ans的值进行修改**

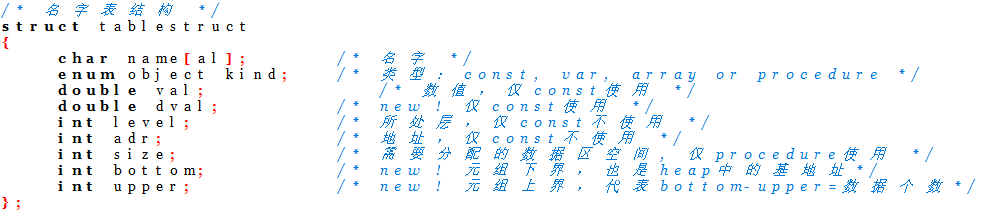
****

FOR语句

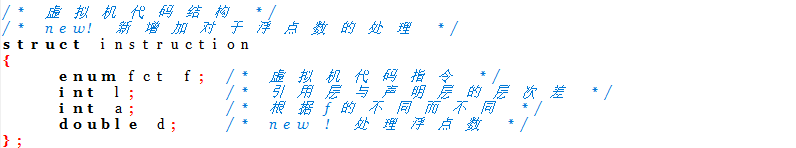
**for语句流程图**

4、各类型运算部分

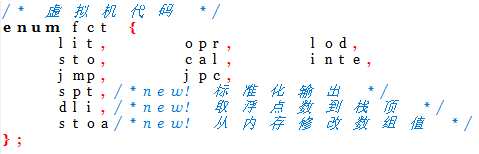
首先由于加入了浮点数操作，需要对于名字表进行扩充，添加浮点数类型变量。



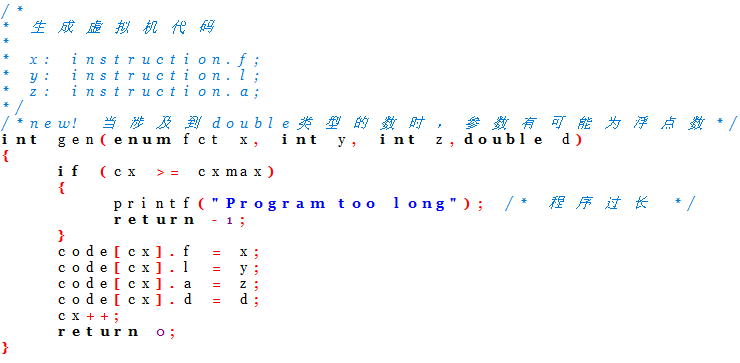
之后需要对于原本的指令结构进行修改，在保证原指令可运行的基础上，添加与浮点数有关的部分，这样就可以对于相同的指令进行重用提高效率。



之后添加与浮点数、数组、元组操作相关的指令

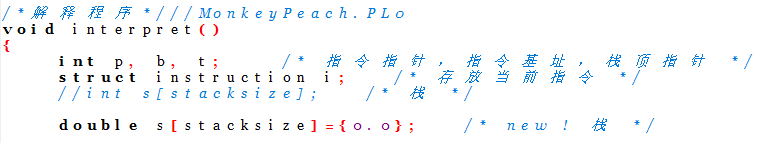


回到编译部分，由于修改了指令的结构，所以需要对于指令生成函数进行修改：

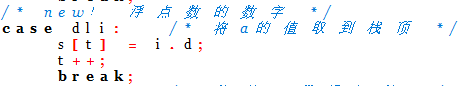


修改过后对于原本的指令需要进行扩充，比如（OPR，0，0）扩充为（OPR，0，0，0）增加一个参数用于传递浮点数。

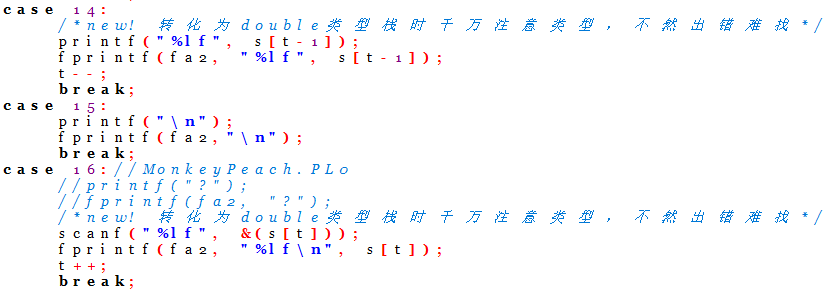
在解释部分进行修改：涉及到的一个关键的修改在于将整型栈修改为浮点型栈。



取浮点数到栈顶：



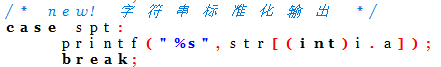
修改输入、输出操作：这里一定要确保类型对应，我在debug期间由于输出类型不对应导致错误很长一段时间未能被发现。



对于元组值的修改涉及到内存（额外开辟的堆空间）的直接修改：

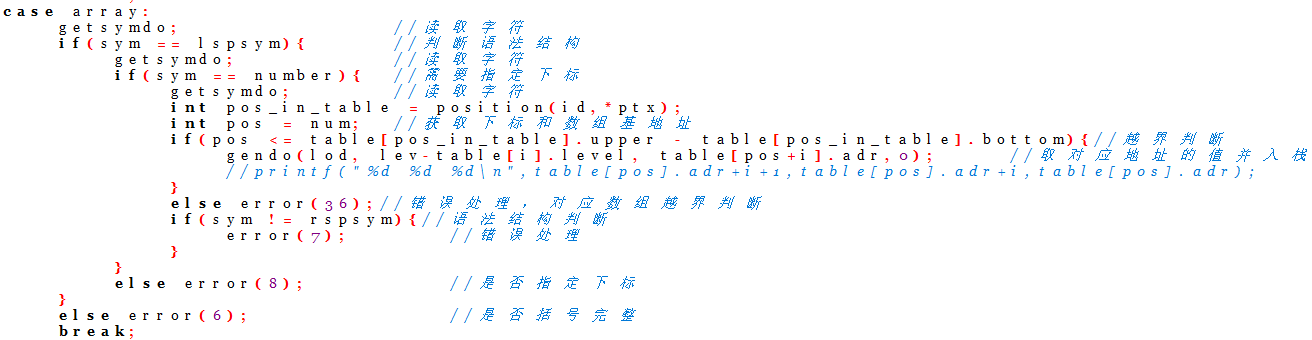


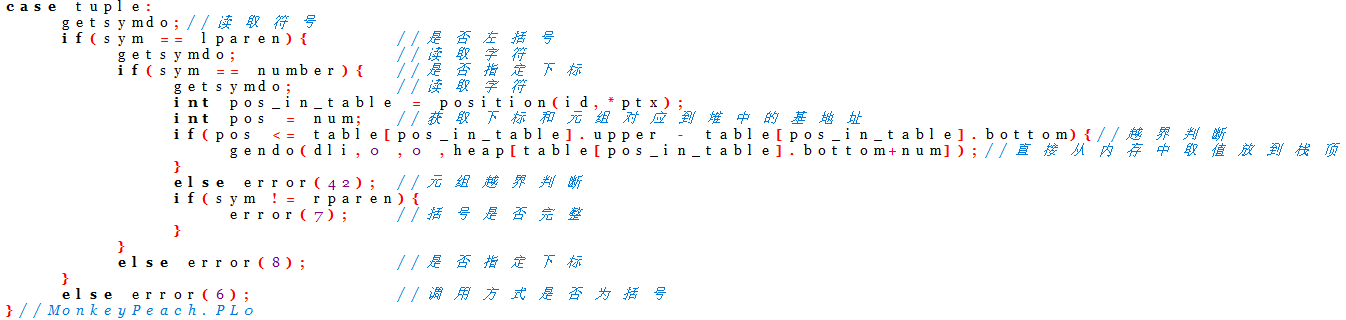
对于引用部分的输出需要结合标准化输出缓冲区：

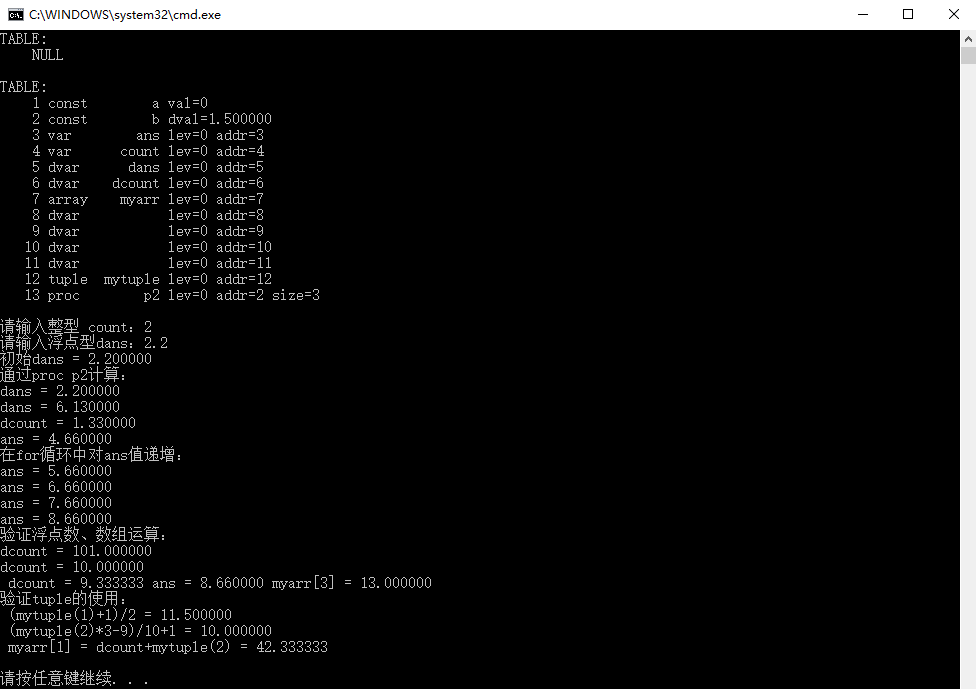


最后修改expression()部分，层层向下分析直到factor()部分，修改case情况，增加各类数据类型的判断：

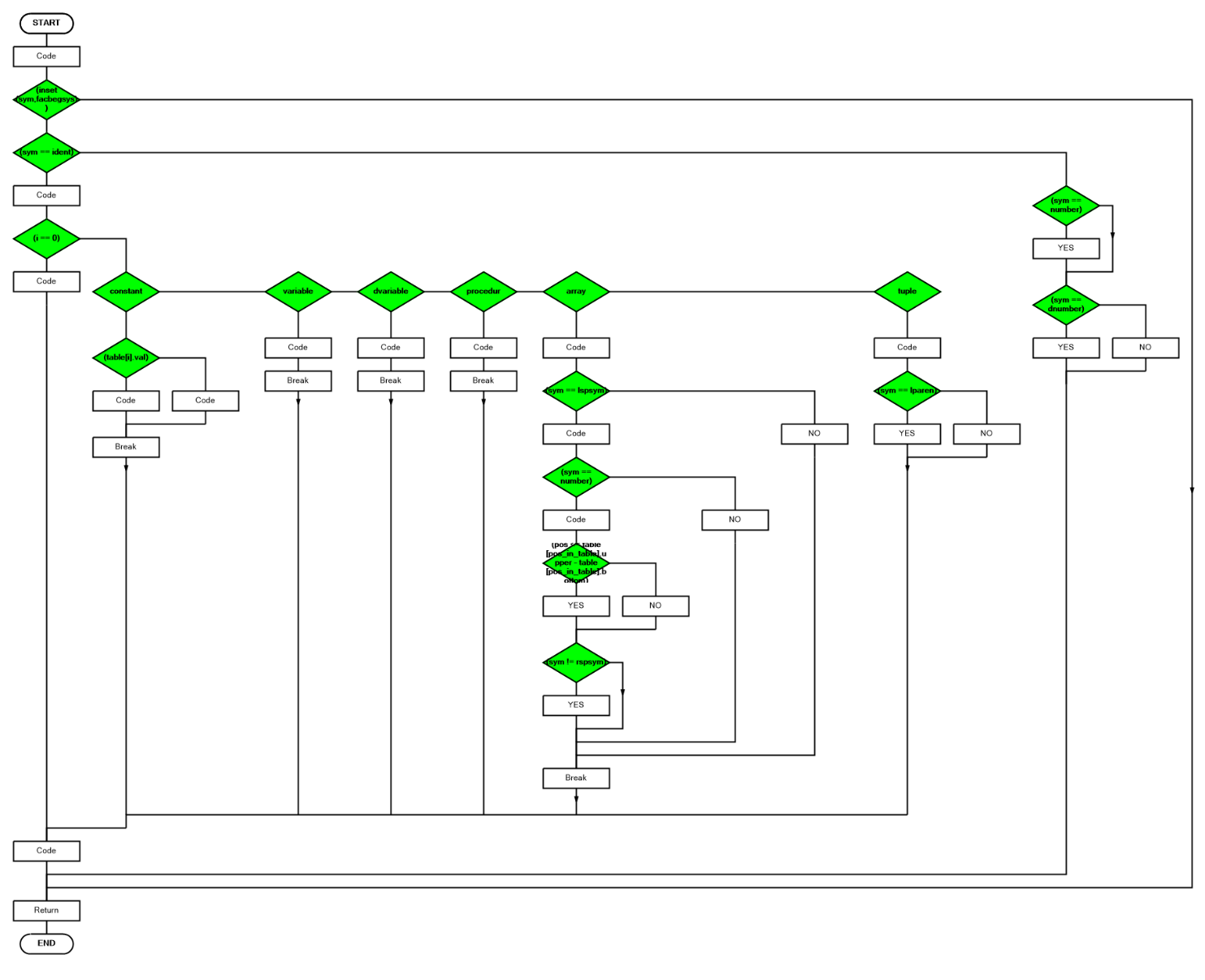








**var、dvar、array[n]、tuple(n)混合四则运算**



常量浮点型

TUPLE类型

ARRA类型

浮点类型

**Factor中涉及到扩充变量的处理流程**

5、错误处理部分

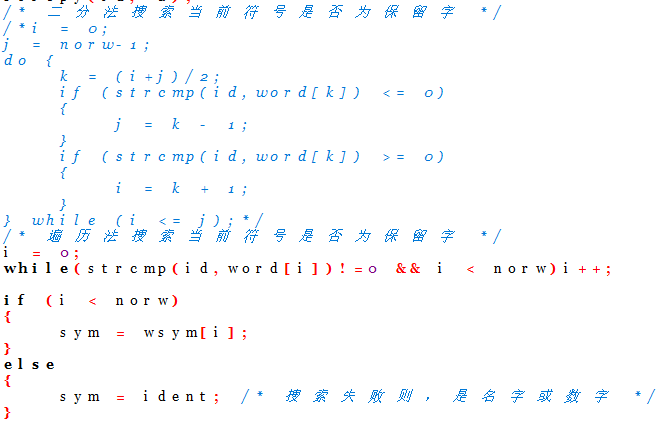
总共分为41种错误，囊括了当前包含的语法错误提示、声明错误提示等，其中LINE\_NUM为全局变量用于保存当前分析到的行数，在进行分析时可以结合对于错误所在的行号进行迅速查找。便于对错误的快速定位。



**错误处理流程**

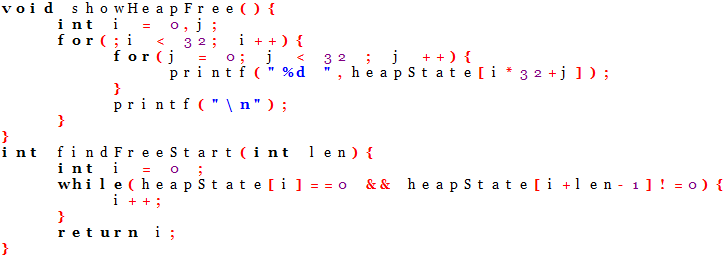
6、其他修改部分

由于需要添加多个关键词，考虑到初始化部分对于两个数组之间的映射关系不便调整，所以我采用简单的遍历方式确定是否为保留字，若要沿用二分法搜索，则需要对于当前的44个符号进行字母序排序，不便于随后的添加和删除。但是采用遍历的方法肯定会一定程度上降低效率。



**其他增加补充部分1**

涉及到元组的空间分配问题，本实验新增两个函数：

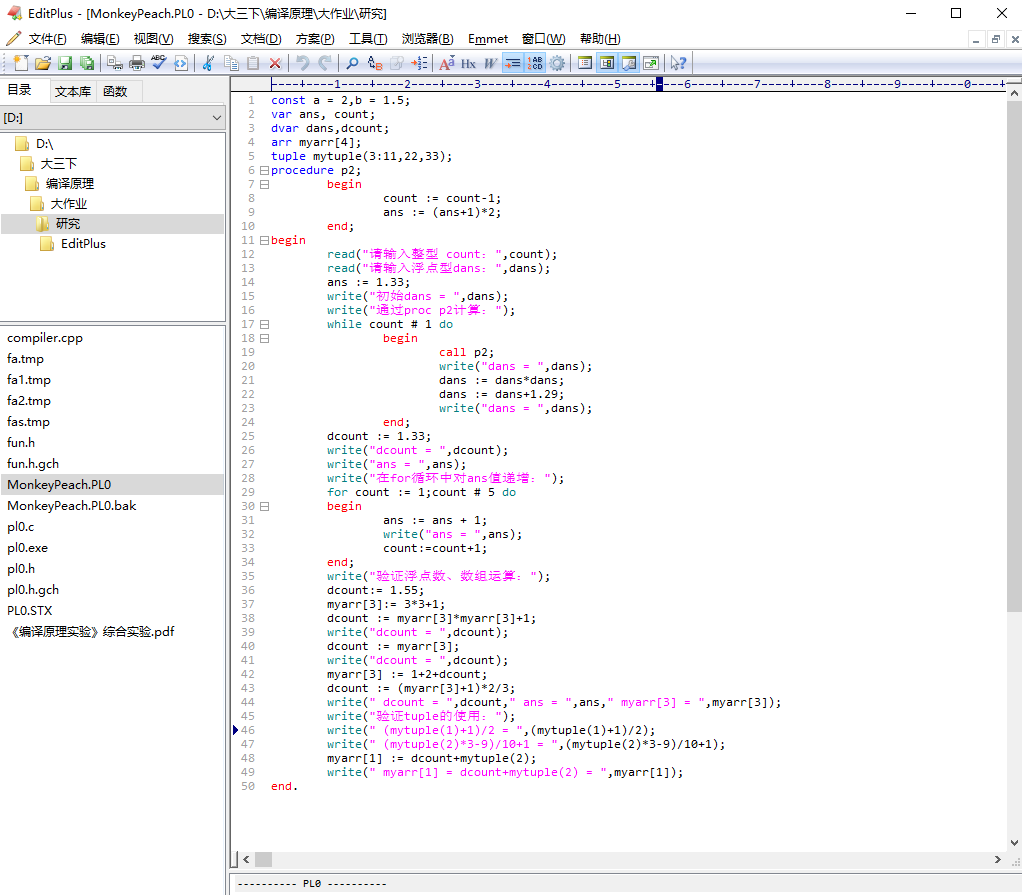


**其他增加补充部分2**

()用于显示当前heap中所有空间的分配情况；()用于元组初始化的时候在所有的heap空间中寻找到一片连续的空闲空间。这样可以采取最先适配空间分配方式。

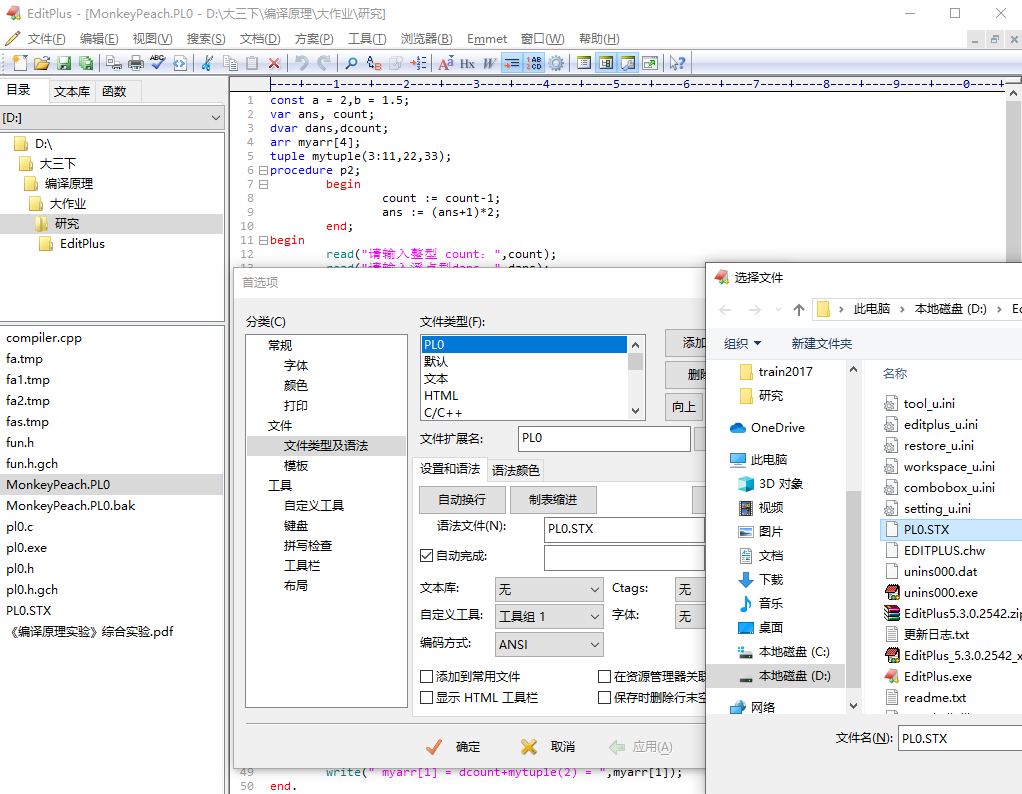
7、配置部分

首先安装。实现代码高亮、代码行数和代码缩进，整体效果如下：



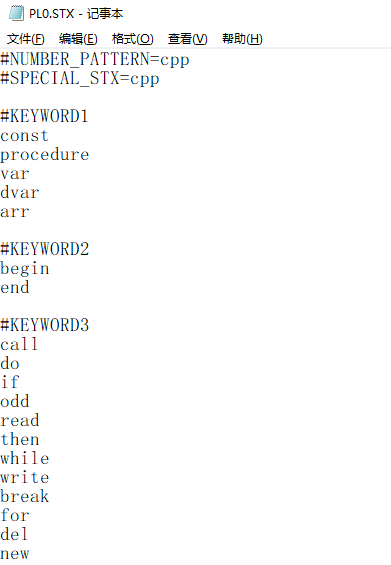
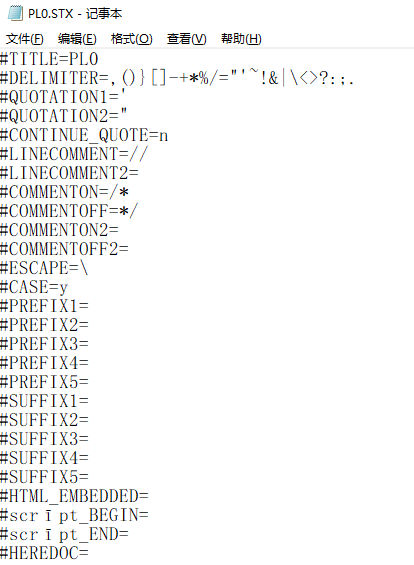
**配置效果总览**

具体STX配置：



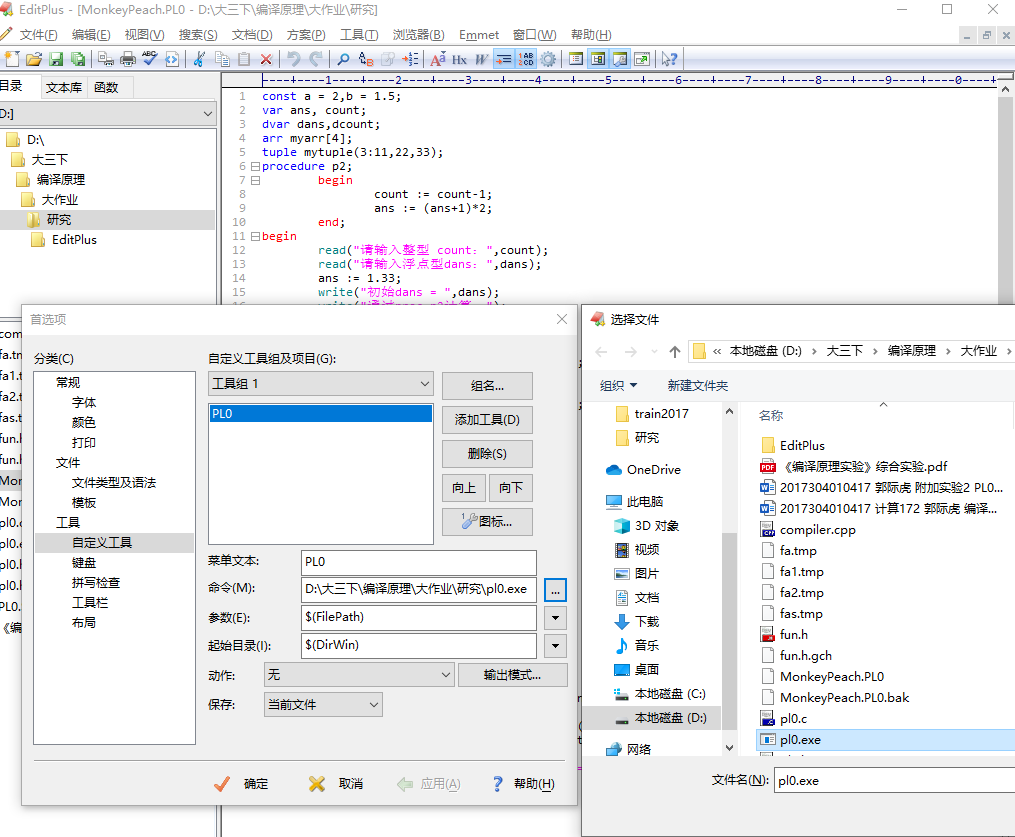
**配置STX文件**

STX内容：



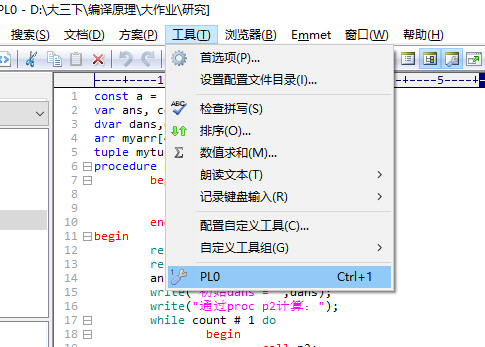
**STX文件内部具体编写**

编译运行配置，由于我没有分离源码中的编译部分和解释部分，所以直接配置相对应的可执行程序即可：



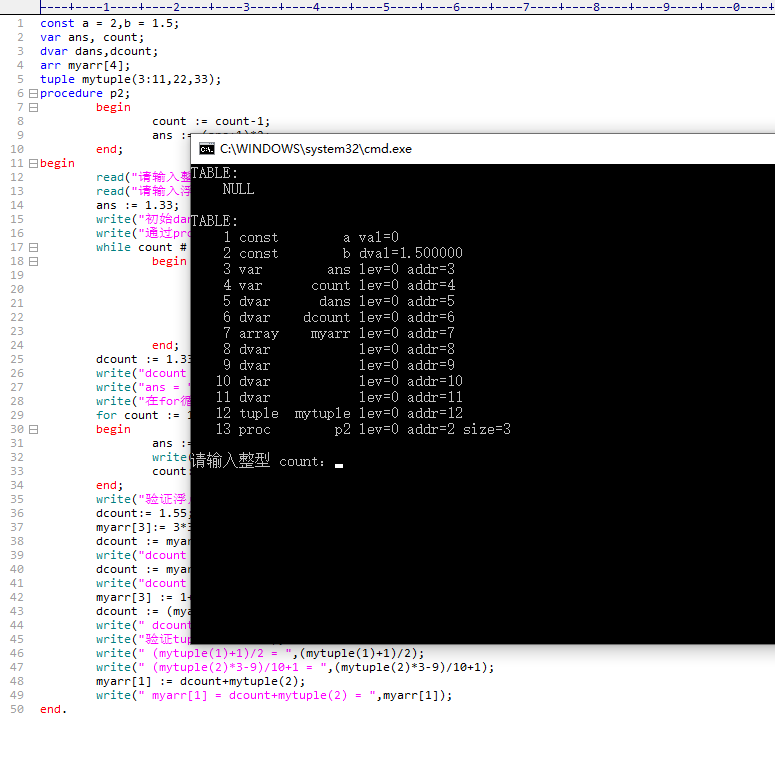
**编译器配置**

其中由于我需要进行输入操作，所以在动作栏我选择无，此时会直接运行我在当前界面写入的代码（需要放在同一文件夹下并确保文件名称正确）。



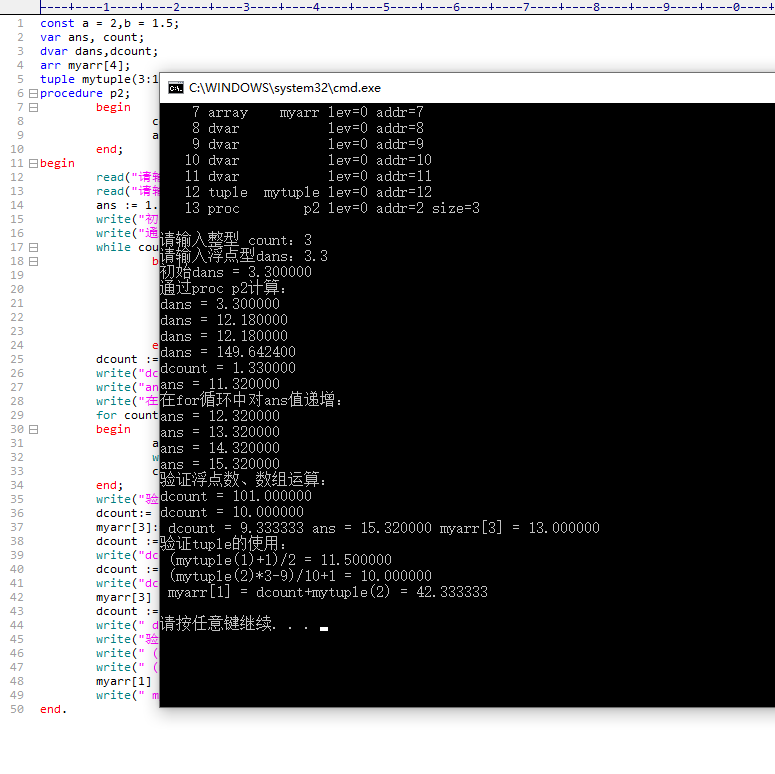
**操作流程1**

点击PL0或者按下快捷键Ctrl+1：弹出控制台界面，类似于Dev C++。



**按照提示输入变量**

按照提示输入整型变量和浮点变量得到结果：



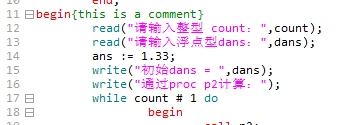
**得到最终结果**

**六、心得与思考**

这次实验是对PL0源码进行扩充。早在开学初期我为了更好的理解老师上课讲的内容就已经对于源码有了一定的阅读。对于一些函数的功能，参数的设置，功能的调用都有了一定的理解。

但是当我真正的自己去尝试修改时还是遇到了不少意外。我在构想初期实际上是还要包括两个功能：对于元组的del和new。分别对应了元组的空间分配和空间删除，我原本预想在此基础之上可以增加一项：对于空间分配的优化。但是当我实际操作起来之后感觉做到的程度都不能令自己满意。其中涉及到的相关代码我都保存在了源码中，只不过是当时一下午的构思形成的代码，可能欠缺一些妥当性。我之所以认为自己写的这两部分代码不符合自己的预期并且没有将之加入到我的实验报告中是因为，我感觉自己在程序中分配空间，又让代码去使用自己分配的空间有点多此一举。我在上学期的操作系统课上也有过类似的想法。这样的操作完全可以通过C、C++底层的设计自动完成。这个想法困扰了我很久，所以最终我也没有将这两个功能纳入自己的实验工作中。

除此之外也遇到了一些有意思的错误：比如在处理注释语句的时候，会报出意想不到的错误：假如我在此处加入一个注释（我在中配置注释语句为{}包括的代码段）



报出的错误竟然是：



而找到我的错误处理函数，其中del后缺少标识符是del tuple的时候才用到，而我在代码中根本没有用到del关键字。这个错误我目前还没有找到相应的解决办法，我的思路是修改词法分析程序，读取字符当遇到注释起始符号 { 时就开始继续推进字符，直到读取到字符 } 时停止，然后再次调用词法分析程序进行分析。

除此之外，对于栈的修改我也下了一番功夫。在一开始我有考虑过使用双栈，但是双栈的话可能会造成虚拟机指令的复杂化，比如我原本只需要从栈指定的地址取数放入栈顶，加入双栈之后可能就需要先从指令栈获取对应地址，通过地址找到对应的整型栈或者浮点型栈中的数据，再将数据放回指令栈的栈顶，这样的话就会增加指令。考虑到对于地址的选择皆为整型而数据可能存在浮点型，所以我对指令做出了一点修改，增加了一个额外的参数实现指令中浮点类型数据的传递，并且修改栈为浮点栈，这样是会减轻工作量，但是也会造成一定的不安全因素。如果时间充沛的话我想还是会选择用双栈来实现的。

另外涉及到浮点栈的修改，期间我还遇到了诸如指令地址错乱、输入输出格式混乱等错误，可见每一次的修改都要经过反复的推敲、反复的修改才能最终成功。一开始我也花了一定的时间去读懂PL0源文件中的虚拟机指令，也正是读懂了那些指令，懂得了每一层在运行初始阶段会先在栈内开辟与符号表中对应变量所需内存相符合大小的栈空间，我才能在后续的浮点型数据操作中萌生自己添加新指令的想法，不然我可能就囿于原本的8条指令无法进一步实现其他功能了。

这次试验之后我对于编译的基本原理，诸如词法分析、语法分析及其底层的实现有了更进一步的理解，尤其是在对词法分析函数修改时，我常常犯多取字符或者少取字符导致编译报错的情况。如果想要扩充源码，就必须更深入的理解，就必须了解每一个模块的功能和结构。在表达式求值时，我对于factor函数的修改有时候会忘记判断地址的偏移，错误的把字符表内位置的偏移当作内存中数据存放的地址的偏移，将偏移量写反、写到括号外等等错误，也让我深刻的意识到了编译原理是一门细节决定成败的科目。

这次的实验我没有完全实现自己预想中的所有工作，如果在考试周结束后有机会，我会抽时间回头再来看看自己当初没有设计好的功能，比如tuple的new和delete功能，甚至比较简单，但是我却遇到了无法处理的错误的注释功能，还有就是IDLE的配置并没有做的很好很完整，比如编译器和解释器没有分开，没有配置ACP文件对语法进行自动补充设置，关于IDLE配置这点是受了前几日汇报的同学的启发，感觉可以尝试一下。这学期的学习给了我很大的帮助，编译原理实验这门课令我受益匪浅。