****



# 编译原理实践

——PL/0编译系统

**姓名： 朱 辉**

**学号：15211041**

**学院：软件学院**

目录

[编译原理实践 I](#_Toc502486330)

[引言 1](#_Toc502486331)

[一、PL/0语言 1](#_Toc502486332)

[1.1 PL/0描述 1](#_Toc502486333)

[1.2 PL/0语言文法 1](#_Toc502486334)

[1.3 PL/0编译系统结构 2](#_Toc502486335)

[二、分析与实现 3](#_Toc502486336)

[2.1 PL/0的词法分析 3](#_Toc502486337)

[2.2 PL/0的符号表管理 4](#_Toc502486338)

[2.2.1符号表结构 4](#_Toc502486339)

[2.2.2符号表管理 4](#_Toc502486340)

[2.3 PL/0的语法分析 4](#_Toc502486341)

[2.4 生成目标代码 5](#_Toc502486342)

[2.4.1 地址返填技术 6](#_Toc502486343)

[2.4.2生成Pcode 6](#_Toc502486344)

[2.5错误处理 7](#_Toc502486345)

[2.5.1错误处理的原则 7](#_Toc502486346)

[2.5.2错误校正 7](#_Toc502486347)

[2.5.3错误信息 7](#_Toc502486348)

[三、扩展：解释器与GUI界面 8](#_Toc502486349)

[3.1 Pcode解释器 8](#_Toc502486350)

[3.2 GUI界面 9](#_Toc502486351)

[3.3 测试样例说明 12](#_Toc502486352)

[3.4 编译系统文件说明 12](#_Toc502486353)

[四、实践感想 12](#_Toc502486354)

[五、参考资料 13](#_Toc502486355)

引言

PL/0编译系统，语言：C++，MFC界面，IDE：VS2015。编译系统包含词法分析、语法分析、符号表管理、语义分析及目标代码Pcode的生成、错误处理以及Pcode的解释执行。整个编译器以语法分析程序为中心，调用词法分析程序识别单词，生成相应的代码，查询或填写相应的符号表项，在出错时调用错误处理程序，报告错误，同时跳过出错部分继续进行语法分析。整个工程耗时两星期，基本达到课程要求。

一、PL/0语言

## 1.1 PL/0描述

PL0是一种十分简单的“高级”程序设计语言，它只有整数一种类型，但却具有相当完全的可嵌套型的分程序结构。PL/0可进行常量定义、变量说明和过程调用，并具有通常程序设计语言所必需的的最基本的语句，如赋值语句、条件语句、循环语句、过程调用语句和复合语句。考虑到输入/输出的需要，我们添加了简单的读、写语句。PL/0过程没有参数，但可以递归调用，因此，过程所加工的数据只能通过全局变量进行传递。

PL/0语言是一种类PASCAL语言，是教学用程序设计语言，它比PASCAL语言简单，作了一些限制。比如：数据类型只有无符号整数；标识符类型只有简单变量（var）和常量（const）；嵌套的层数最大值为3等。

## 1.2 PL/0语言文法

<程序> ::= <分程序>.

<分程序> ::= [<常量说明部分>][变量说明部分>][<过程说明部分>]<语句>

<常量说明部分> ::= const<常量定义>{,<常量定义>};

<常量定义> ::= <标识符>=<无符号整数>

<无符号整数> ::= <数字>{<数字>}

<标识符> ::= <字母>{<字母>|<数字>}

<变量说明部分>::= var<标识符>{,<标识符>};

<过程说明部分> ::= <过程首部><分程序>;{<过程说明部分>}

<过程首部> ::= procedure<标识符>;

<语句> ::= <赋值语句>|<条件语句>|<当型循环语句>|<过程调用语句>|<读语句>|<写语句>|<复合语句>|<重复语句>|<空>

<赋值语句> ::= <标识符>:=<表达式>

<表达式> ::= [+|-]<项>{<加法运算符><项>}

<项> ::= <因子>{<乘法运算符><因子>}

<因子> ::= <标识符>|<无符号整数>|'('<表达式>')'

<加法运算符> ::= +|-

<乘法运算符> ::= \*|/

<条件> ::= <表达式><关系运算符><表达式>|odd<表达式>

<关系运算符> ::= =|<>|<|<=|>|>=

<条件语句> ::= if<条件>then<语句>[else<语句>]

<当型循环语句> ::= while<条件>do<语句>

<过程调用语句> ::= call<标识符>

<复合语句> ::= begin<语句>{;<语句>}end

<重复语句> ::= repeat<语句>{;<语句>}until<条件>

<读语句> ::= read'('<标识符>{,<标识符>}')'

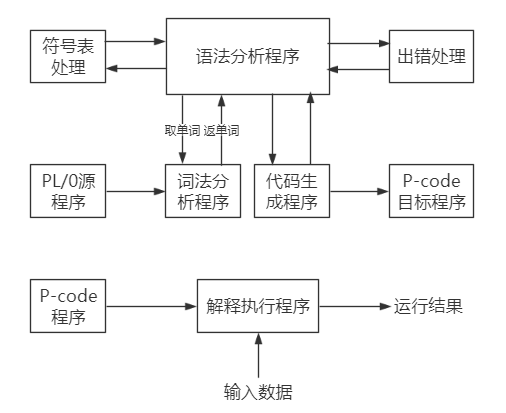
<写语句> ::= write'('<标识符>{,<标识符>}')'

<字母> ::= a|b|...|X|Y|Z

<数字> ::= 0|1|2|...|8|9

## 1.3 PL/0编译系统结构

PL/0编译系统是一个编译-解释执行程序，整个编译过程分两个阶段进行。第一阶段先把PL/0源程序编译成假想计算机的目标程序（P-code指令），第二阶段再对该目标程序进行解释执行，得到运行结果。PL/0编译程序采用一遍扫描，即以语法分析为核心，由它调用词法分析程序取单词，在语法分析过程中同时进行语义分析处理，并生成目标指令。如遇到语法、语义错误，则随时挑用出错处理程序，打印出错信息。在编译过程中要利用符号表的登录和查找来进行信息之间的联系。一边扫描的PL/0编译和P-code解释执行框图如图1所示。



**图1 PL/0编译系统结构框图**

二、分析与实现

## 2.1 PL/0的词法分析

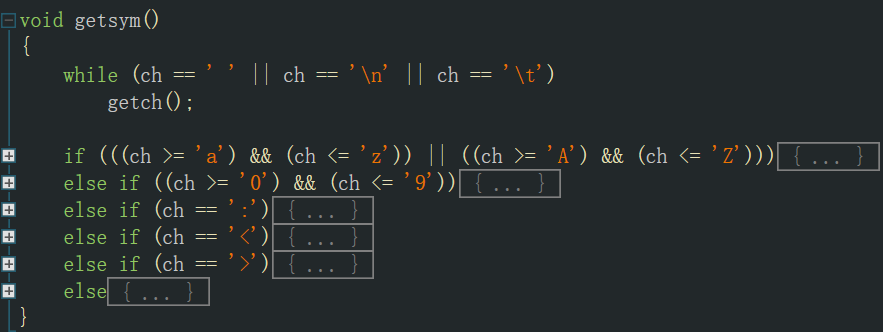
PL/0编译系统中所有的字符和字符串的类型可简单分为七类，为如下表格：

**表1 PL/0编译系统字符/字符串类型**

|  |  |
| --- | --- |
| 保留字 | Begin,end,if,then,else,const,procedure,var,  do,while,odd,call,read,write,repeat,until |
| 算数运算符 | + | - | \* | / |
| 比较运算符 | <> | < | <= | > | >= | = |
| 赋值符 | := | = |
| 标识符 | 变量名，过程名，常数名 |
| 常数 | 整数 |
| 界符 | , | . | ; | ( | ) |

PL/0的词法分析程序getsym作为一个独立的子程序有语法分析程序调用，它的主要功能及代码结构如下：

1. 跳过源程序中的空格、缩进、换行等字符。
2. 识别单词符号，并把其类别以枚举变量值存入sym中。
3. 用变量id存放标识符，二分法查找保留字数组word[]。
4. 若读取无符号整数，则将相应数字自妇产转换为整数存于变量num中国。
5. 通过调用getch取得一个字符，为了优化读取字符效率，每次读取一行源程序，存入缓冲区line，变量ll为源程序当前行的长度，cc为当前正在读取的字符位置。



**图2 词法分析getsym函数**

PL/0的语法分析采用了“单符号先行”技术，即在进入某个语法成分的分析子程序之前，先读取一个单词放入sym中，相对应的，在getsym中，也总是先读取一个字符存放在变量ch中。整个编译过程是先读一个单词加一个字符。

## 2.2 PL/0的符号表管理

### 2.2.1符号表结构

符号表数组table存放所有的符号表项，其中符号表项结构体定义如下：

struct tab {

string name;//名字

objecttype kind;//类型（const/var/proc）

int val;//值，const专用

int level;//层次，var/proc专用

int adr;//地址，var/proc专用

};//符号表项

### 2.2.2符号表管理



**图3 PL/0符号表管理函数**

1. 登录符号表，即新增符号表项。

k:标识符的种类，可以为const，var或procedure

dx:当前应分配的变量的相对地址，分配后要增加1

lev:标识符所在的层次

tx:符号表尾指针的指针，可以直接改变符号表尾指针的值

1. 查询符号表，查找标识符在符号表中的位置从tx开始倒序查找标识符。

找到则返回在符号表中的位置，否则返回0。

id:要查找的标识符名字

tx:当前符号表尾指针（倒序查找）

## PL/0的语法分析

PL/0采用了**递归子程序法**进行语法分析，即为每一个语法成分编写一个分析子程序，根据当前读取的符号，可以选择相应的子程序进行语法分析。采用不带回溯的递归子程序法，对语言文法有一定的要求：

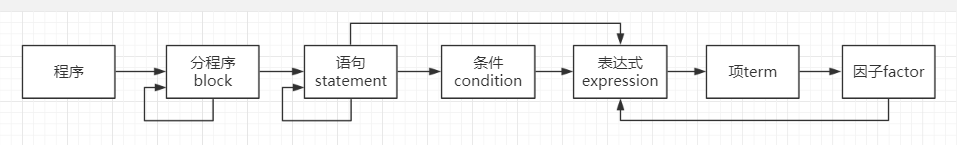
1. 文法必须是非左递归；
2. 文法的任一非终结符，其规则右部多个选择所生成的FIRST集合两两不相交；
3. 若文法具有形如，则。

根据PL/0的文法BNF表达，下表中列出了PL/0有关的语法成分的FIRST集合FOLLOW集，可以判断出满足递归子程序的要求。表说明：R表示关系运算符，id和num分别表示标识符和无符号整数。

**表2 PL/0文法非终结符的开始符号集与后继符号集**

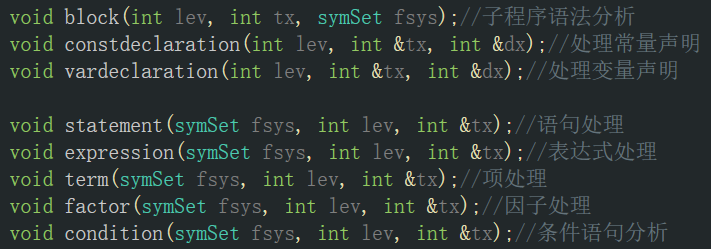
|  |  |  |
| --- | --- | --- |
| 非终结符 | FIRST集 | FOLLOW集 |
| 分程序block | const var procedure id if call begin while repeat read write | . ; |
| 语句statement | Id call begin if while read write until | . ; end |
| 条件condition | odd + - ( id num | then do |
| 表达式expression | + - ( id num | , ; ) R end then do |
| 项term | id num ( | , ; ) R + - end then do |
| 因子factor | id num ( | . ; ) R + - \* / end then do |

PL/0的语法关系调用图如下，可以清晰地看出进行直接或间接的递归调用。



**图4 PL/0语法关系调用图**

与上图相对应，图5是语法分析函数声明，具体实现可查看syntax.cpp。



**图5 PL/0语法关系函数声明**

## 2.4 生成目标代码

Pcode语言：一种栈式机的语言。此类栈式机没有累加器和通用寄存器，有一个栈式存储器，有四个控制寄存器（指令寄存器 I，指令地址寄存器 P，栈顶寄存器 T和基址寄存器 B），算术逻辑运算都在栈顶进行。

指令格式形如F L A，其中F表示操作码，可用枚举变量表示；L表示层次差 （标识符引用层减去定义层）；A不同的指令含义不同。指令项结构体定义如下：

struct instruction {

fct f;//操作码

int l;//层次差

int a;

};//操作指令

### 2.4.1 地址返填技术

对于if then [else]，while do和repeat until语句，要生成跳转指令，采用地址返填技术。目标代码生成模式如下：

1. if <condition> then <statement>[else]

<condition>

JPC addr1

<statement>

addr1:[else]

1. while <condition> do <statement>

addr2:<condition>

JPC addr3

<statement>

JPC addr2

addr3:

1. repeat <statement> until <condition>

addr4:<statement>

<condition>

JPC addr4

### 2.4.2生成Pcode

在语法分析的过程中，同时也在不断地生成Pcode。在语法分析函数中通过调用gen函数将每条指令存入code指令数组中，供之后解释执行。Listcode函数输出所有Pcode指令，存入outPcode.txt文件中。



**图6 PL/0 Pcode管理函数**

## 2.5错误处理

### 2.5.1错误处理的原则

1. 任何源程序输入序列都不会导致编译工作的崩溃；
2. 尽可能多地发现源程序中的语法和语义在错误，并尽可能准确指出错误位置和错误属性；
3. 尽可能进行校正，对于出现的错误，予以校正使编译继续进行。

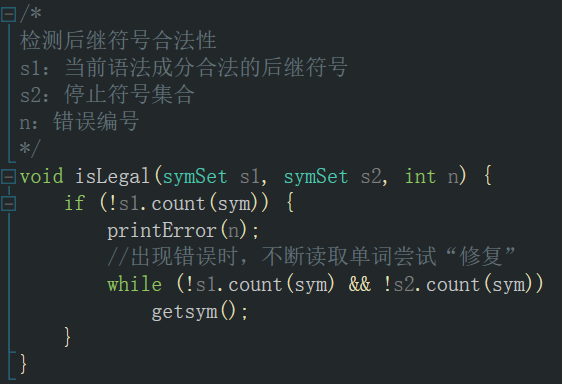
### 2.5.2错误校正

对于简单的错误，如遗忘了分号，赋值符号写错等，除了给出错误信息，可以予以简单校正，使程序尽可能继续进行。

在每个语法分析子程序的入口或出口处，调用isLegal函数, 检测下一个取得的单词是否是当前语法成分的合法头符号集或者后继符号，若不是，给出错误信息，并且跳读一段程序，直至取得合法的后继符号。

然而，为了防止跳读的程序段太多，程序中设置了一个停止符号集合，此集合中元素为明显可以使程序开始正常编译的符号，所以只要新取来的符号属于合法后继符号集或者停止符号集，都可以停止跳读。

在block的出口处，statement的出口处，factor的入口和出口处，都调用了合法性检测函数，isLegal函数定义如下：



**图7 PL/0语义合法性判断**

### 2.5.3错误信息

教材中给出了26类错误信息，个人修改了这些错误信息的描述，使之更加清晰易懂，之后又添加了几类其他的错误信息，尽可能地检测出源程序中存在的问题。

**表3 PL/0编译程序的错误信息**

|  |  |
| --- | --- |
| 出错编号 | 出错原因 |
| 1 | 常数说明中应是'='而不是':=' |
| 2 | 常数说明中'='后应为数字 |
| 3 | 常数说明中标识符后应为'=' |
| 4 | const,var,procedure后应为标识符 |
| 5 | 漏掉逗号或分号 |
| 6 | 过程说明后的符号不正确 |
| 7 | 应为语句开始符号 |
| 8 | 程序体内语句部分后的符号不正确 |
| 9 | 程序结尾应为句号 |
| 10 | 语句之间漏了分号 |
| 11 | 标识符未说明 |
| 12 | 不可向常量或过程赋值 |
| 13 | 赋值语句中应为赋值运算符':=' |
| 14 | call后应为标识符 |
| 15 | call后标识符属性应为过程,不可调用常量或变量 |
| 16 | 条件语句中缺失then |
| 17 | 应为分号或end |
| 18 | while型循环语句中缺失do |
| 19 | 语句后的符号不正确 |
| 20 | 应为关系运算符 |
| 21 | 表达式内不可有过程标识符 |
| 22 | 缺失右括号 |
| 23 | 因子后不可为此符号 |
| 24 | 表达式不能以此符号开始 |
| 25 | repeat循环语句中缺失until |
| 26 | 代码太长，无法编译 |
| 27 | RuntimeError，地址偏移越界 |
| 28 | Read语句括号内不是标识符 |
| 30 | 这个数太大，超过INT32\_MAX |
| 40 | 缺失左括号 |

三、扩展：解释器与GUI界面

## 3.1 Pcode解释器

其实在生成Pcode时，已经理解了Pcode每条指令的含义，很快就可以写出相应的解释器，代码对应于interpret.h和interpret.cpp中。

假想的一个PL/0计算机，有两个存储器、一个指令寄存器和三个地址寄存器组成。存储器code存放程序Pcode指令，存储器S作为运行栈， 实现栈式动态存储分配。指令寄存器ins存放当前执行的Pcode指令；栈顶地址寄存器top作为栈顶指针，总是指向运行栈S的栈顶；程序地址寄存器p存放下一条要执行的指令地址（数组索引）；基地址寄存器bp，存放当前运行的分程序数据区在数据栈S中的起始地址，即bp总是指向动态链的链头。

如下表格4是Pcode的指令含义及其解释执行过程。说明：base函数作用是通过静态链求出数据区的基地址，Cin、Cout为输入输出流，对应着input.txt与output.txt文件。

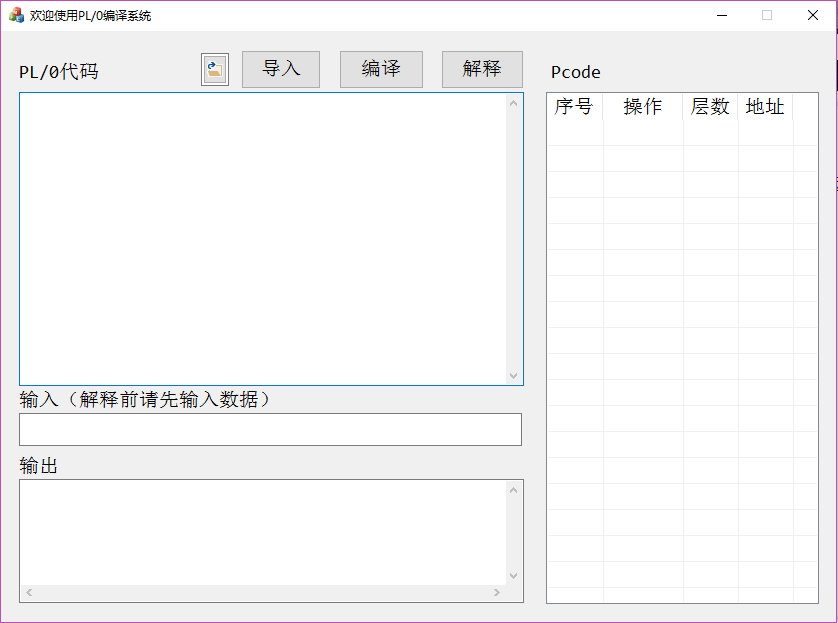
**表4 Pcode的含义与解释执行**

|  |  |  |
| --- | --- | --- |
| 指令 | 具体含义 | 具体操作 |
| LIT 0 a | 取常量a置于栈顶 | top++;  S[top] = ins.a; |
| OPR 0 a | 执行运算，a的值表示执行何种运算 | 根据a值的不同执行不同的操作 |
| LOD l a | 取变量（层次差为l，相对地址为a）置于栈顶 | top++;  S[top] = S[base(ins.l) + ins.a]; |
| STO l a | 将栈顶值存入变量（层次差为l，相对地址为a） | S[base(ins.l) + ins.a] = S[top];  top--; |
| CAL l a | 调用过程（层次差为l，入口指令地址为a） | S[top + 1] = base(ins.l);  S[top + 2] = bp; S[top + 3] = p;  bp = top + 1; p = ins.a; |
| INT 0 a | 分配空间，栈指针top增加a | top += ins.a; |
| JMP 0 a | 无条件跳转至地址a | p = ins.a; |
| JPC 0 a | 条件跳转至地址a | if (S[top] == 0) p = ins.a;  top--; |
| RED l a | 读数据，存入变量（层次差为l，相对地址为a） | Cin >> S[base(ins.l) + ins.a]; |
| WRT 0 0 | 写数据，将栈顶值输出 | Cout << S[top] << endl;  top++; |

## 3.2 GUI界面

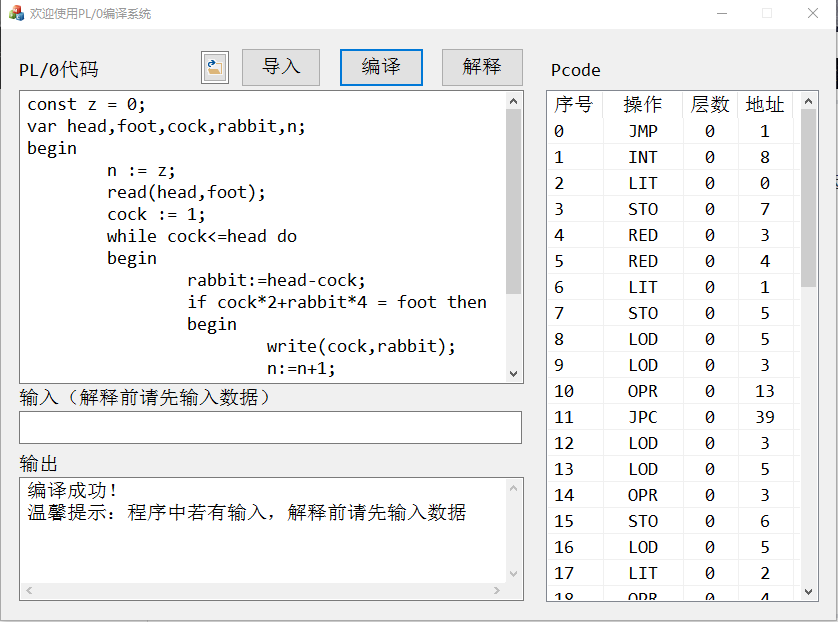
PL/0编译器界面采用简易MFC制作，编译系统初始界面如图8。

界面中左上文本框为代码输入区，可直接输入Pl/0代码。上面有三个按钮，其中**导入按钮**可以选择文件导入代码编辑区；点击**编译按钮**，将会编译代码编辑区中的代码，右侧显示Pcode，左下方会显示编译成功或者显示错误信息，其中代码编辑区的代码会保存在inCode.txt中，输出Pcode会保存在outPcode.txt中；在点击**解释按钮**前，如果程序中有输入数据，先在输入框中输入数据（数据可多不可少），再点击解释按钮，否则会弹出数据不够，左下的输出框会显示解释结果。所有重要的信息在后台都有文件的备份。



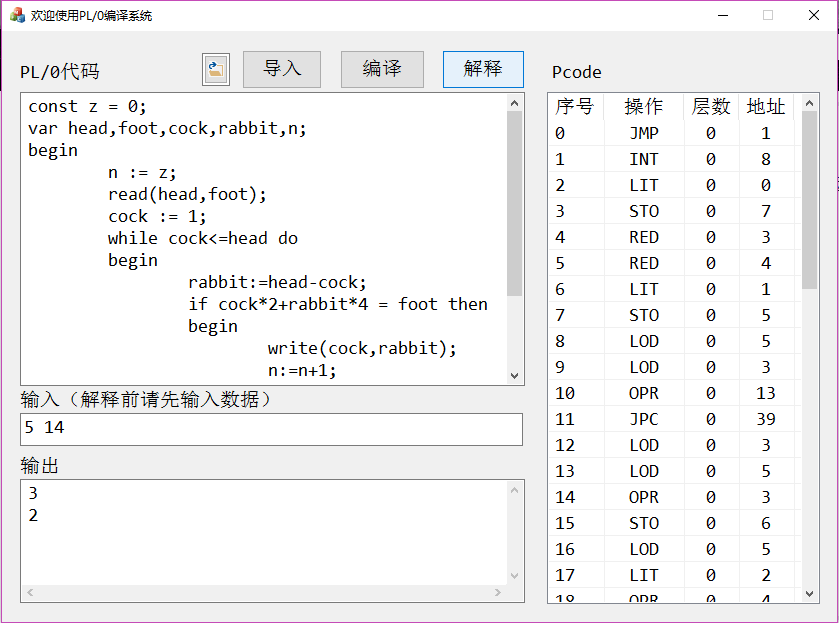
**图8 MFC初始界面**

这里，我们导入测试样例1求解鸡兔同笼问题（吐槽一句，这个求解代码方法错误），图9是点击编译之后的结果。



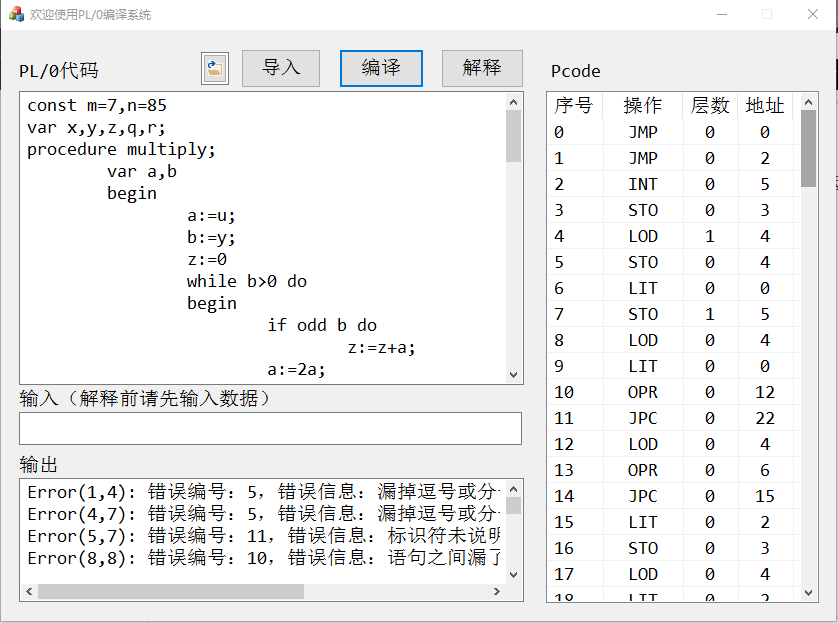
**图9 MFC编译代码界面**

我们现在输入框中输入数据，比如5 14，在点击解释按钮，在左下的输出框中就能看到程序执行的结果啦！



**图10 MFC执行代码界面**

再来看一个输出错误信息的样例， 导入样例4语法错误样例，再点击编译，结果如图11，虽然出现了很多很多条错误信息（必须的），在右侧还是可以看到100多条Pcode指令，说明我们的错误校正还是挺有用的，至少没有直接结束编译，



**图11 MFC执行代码界面**

## 3.3 测试样例说明

一共设计七个测试样例文件，如下表格

**表5 PL/0编译系统测试用例**

|  |  |
| --- | --- |
| 样例编号 | 样例描述 |
| 1 | 教材P309例14.1 求解鸡兔同笼问题 |
| 2 | 教材P310例14.2 求最大公约数和最小公倍数 |
| 3 | 教材P310例14.3 打印素数表 |
| 4 | 教材P320例14.7.2 语法错误样例 |
| 5 | if then else测试样例 |
| 6 | repeat until测试样例 |
| 7 | 个人错误测试样例 |

## 3.4 编译系统文件说明

**表6 PL/0编译系统源代码文件**

|  |  |
| --- | --- |
| 文件名 | 文件描述 |
| console.h/console.cpp | 编译系统程序总控程序，初始化变量，并启动词法、语法分析 |
| error.h/error.cpp | 错误输出，分析过程中随时输出错误信息 |
| Init.h | 编译系统各部分“联系人”，保存着所有的全局变量 |
| interpret.h/interpret.cpp | 解释执行模块，包含base函数与interpret函数 |
| lexica.h/lexcica.cpp | 词法分析模块，包含getch、getsys、isLegal函数 |
| pcode.h/pcode.cpp | Pcode处理模块，包含gen、listcode函数 |
| sytax.h/sytax.cpp | 语法分析模块，包含递归子程序各个函数 |
| sysTab.h/sysTab.cpp | 符号表管理模块，包含enter、position函数 |
| inCode.txt | 代码输入区的备份 |
| input.txt | 数去数据的备份 |
| outError.txt | 错误处理信息的备份 |
| outPcode.txt | Pcode指令集的备份 |
| output.txt | 输出数据的备份 |

四、实践感想

不得不说，语言处理是个细腻差事。从最开始根本不懂什么是Pcode，到现在甚至可以大脑生成Pcode（开玩笑的），中间是个“痛苦”的过程。

由于之前对PL/0进行过词法分析，这一块问题倒是不大，有所改变的是这一次并不是整个代码一次全部分析完，而是通过语法分析程序一个一个取得单词，作为语法分析的资源池。最难的一块当属语法分析模块，虽然对递归子程序法比较了解，实验也写过，语句、表达式、条件、项之间的递归调用自然没问题，难就难在语法分析程序是整个编译系统的核心，他另一边还连着符号表以及Pcode的生成，最后还有众多的错误处理，这的是一下子炸开了，删掉重写了好多次，肝了n天。最后慢慢地，一个一个慢慢加上去，而不是想到什么写什么，一下子符号表，一下子又写错误处理，到头来代码乱成一团，自己都看不懂了。

个人认为第二个难点在于Pcode的理解，如果不能理解，不仅无法生成正确的Pcode代码，扩展中的解释执行更是别想了。虽然只有10个不同的指令类别，用起来时还是变换万千的，从网上找了些博客，一条一条地理解每类指令的作用，以及在分析过程中什么时候需要什么指令操作，最后才在语法分析程序中添加了Pcode的生成代码，实在是不容易。在理解了Pcode后，写解释执行的程序也就变得简单了，只需要两个函数，100多行代码就可以模拟栈式计算机解释执行Pcode了。

其实错误处理也是有难度的，因为教材中已经给出了26条错误信息，一开始慢慢的加，总是不能覆盖所有的错误，大概花了两天的时间才把教材中的所有错误都用上了，甚至还自己加了几条新的错误信息，可把我牛逼坏了。

通过这次实践，第一感受就是写代码条例要清晰，头脑要冷静。千万不能想到什么就写什么，到头来真的是鬼也看不懂的代码。功能也是要一个一个地加，写好一个，再来写下一个，不能多路开发。可以说，这个实践作业包括了本学期编译原理课程的大部分重要内容，在将近期末之际，也顺便复习了一波课程内容，一点不亏。结合每一章的重点内容，凑齐了PL/0编译系统的每一个模块，最终效果不错，不负多个晚上的熬夜以及那快掉完的头发sad

有些不满意的就是GUI界面了，一直以来就不太会用C++写界面，MFC更是不太熟练，用起来比较生硬，最后功能虽然比较齐全，外表看起来还是挺朴素的。另外一个不太好的设计就是输入数据那块，要求事先输入数据是个不太好的设计，如果可以在执行过程中弹出窗口进行输入的话或更好一些，这也是需要改进的地方。

不管怎么说，这次编译实践的收获是非常大的，一方面复习了课程内容，整合了几乎所有的知识，另一方面也锻炼了写代码的能力，对于变成有了更深刻的理解。最后，感谢邵兵老师一个学期来的悉心教诲，感谢助教的帮助，辛苦你们了，祝你们2018开开心心，心想事成！

朱辉

2017年12月31日

五、参考资料

[1] 张莉等．编译原理及编译程序构造