

**PL\0编译器文档**

|  |  |
| --- | --- |
| 姓名： | 许鸿智 |
| 专业： | 软件工程 |
| 学院： | 软件学院 |
|  |  |

北京航空航天大学·软件学院

**PL/0编译器文档实验**

**一、实验要求**

* 1. 以个人为单位进行开发，不得多人合作完成。
  2. 共32个学时。利用每周三上课时间和课余自己完成。
  3. 细节要求：

1. 输入：符合PL/0文法的源程序（自己要有5个测试用例，包含出错的情况，还要用老师提供的测试用例进行测试）
2. 输出：P-Code（解释器可选）
3. 错误信息（参见教材第316页表14.4）。
4. P-Code指令集（参见教材第351页表15.14）。
5. 语法分析部分要求统一使用递归下降子程序法实现。
6. 编程语言自定，可使用C、C++、C#或Java等。
7. 上交材料中不但要包括源代码（含注释）和可执行程序，还应有完整文档。

**二、PL/0语言描述**

PL/0型语言是Pascal语言的一个子集。作为一门教学用程序设计语言，它比PASCAL语言简单，并作了一些限制。PL0的程序结构比较完全，相应的选择，不但有常量，变量及过程声明，而且分支和循环结构也是一应俱全。

PL/0文法的EBNF所示：

<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

**三、程序功能及介绍**

该项目使用python语言编写，实现了对PL/0源代码的词法分析，语法分析，语义分析及Pcode代码生成，也编写了Pcode代码解释程序。

**3.1运行方法**

1.安装python 3.x（建议使用python 3.6） 环境.

执行以下指令：

cd core\_code

python main.py

2.安装python 3.x环境及pycharm IDE，使用pycharm打开项目文件中.idea/pl0python.iml文件，运行main.py代码

**3.2 目录结构**

目录结构如下图：

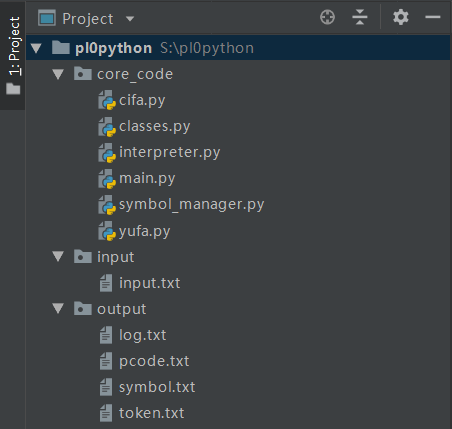


图3.2 目录结构

core\_code目录下为程序核心代码，main.py是主程序入口，cifa.py为词法分析程序，yufa.py中是语法分析、语义分析、pcode生成的代码，classes.py中定义了程序所需要的类（Token、Symbol、Error、Pcode），interpreter.py是解释器，symbol\_manager.py是符号表管理工具。

Input目录下是input.txt，该文件为待编译的pl0源码存放处。

Output目录下是程序运行的各输出文件

**3.2 功能介绍**

主要功能：

将需要编译的pl0代码存放到input/input.txt中 ，运行main.py将会对pl0源码进行一个编译流程。

编译后如果编译成功，输出“Compile succeed.”并将编译生成的token表、符号表、pcode分别输出到output文件夹下的token.txt、symbol.txt、pcode.txt中。且用户可以选择是否调用解释器来运行pl0程序。

如果编译失败，则会输出“Compile failed”，并将错误信息输出到log.txt中。

**3.3 功能演示**

将待编译代码存放于input/input.txt中，运行core\_code/main.py

编译成功则会显示成功信息并将相应的token表，符号表及Pcode表输出到响应文件。

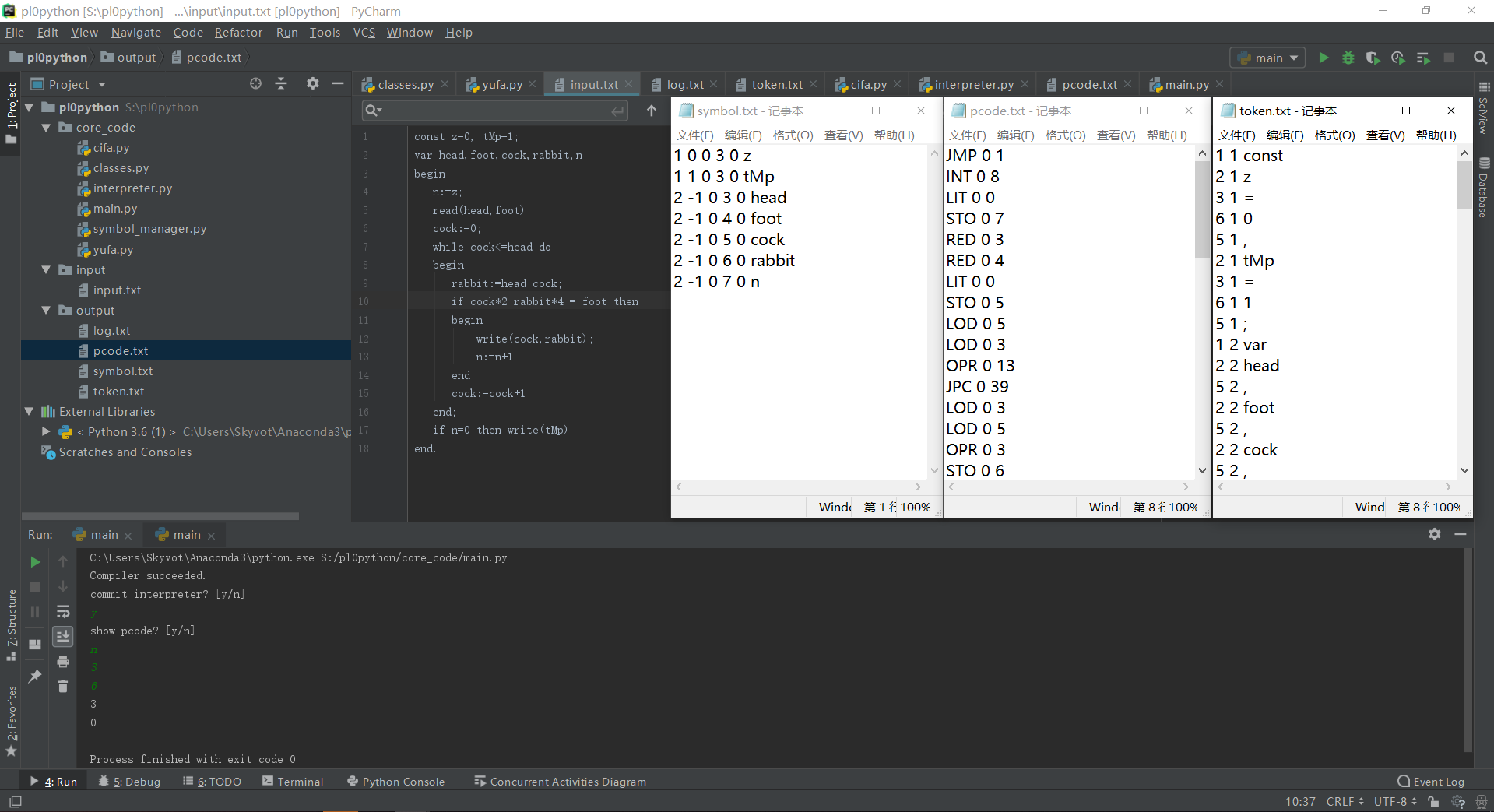


图2 运行成功示意图

编译失败则会给出错误提示，并将编译信息保存到log.txt。

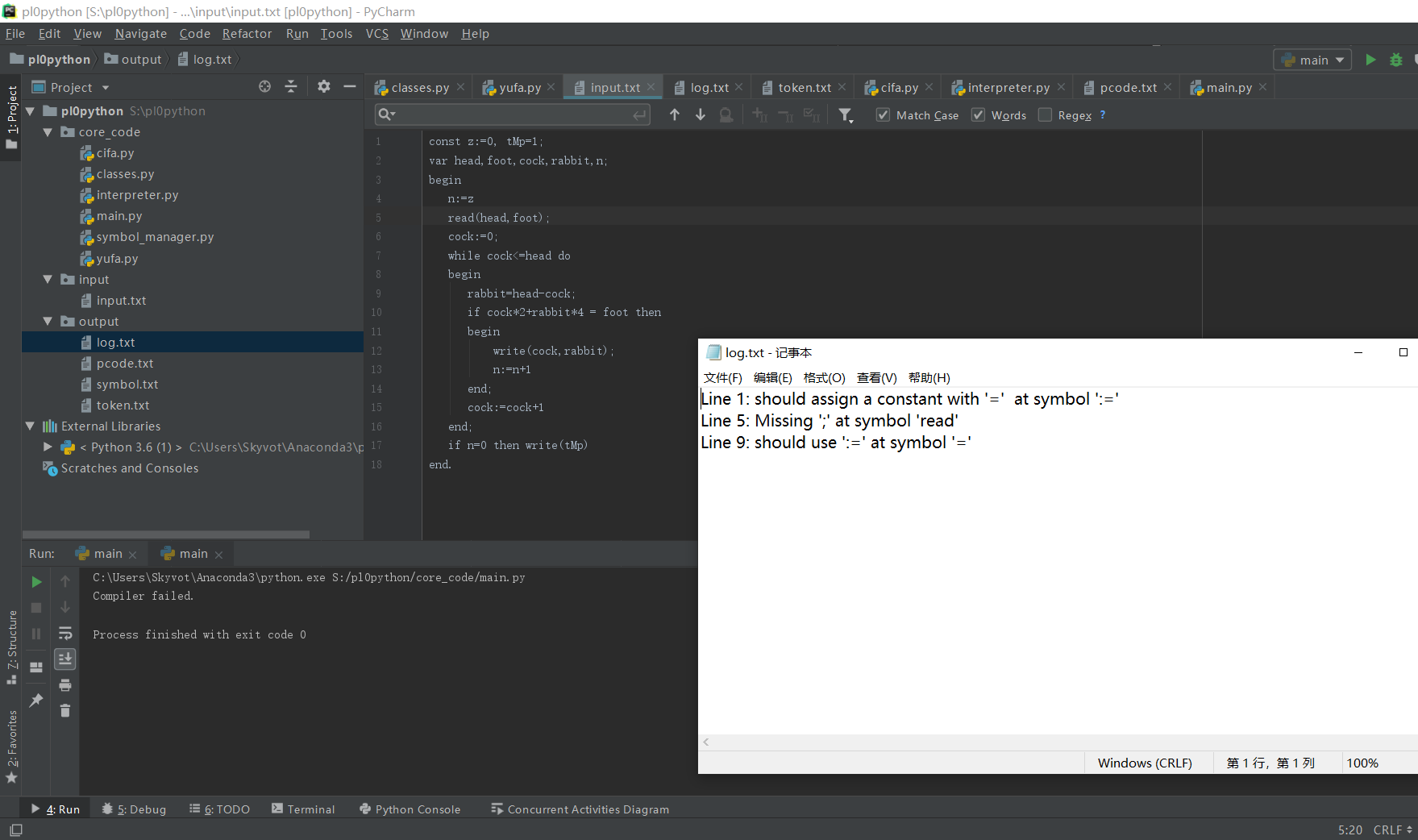


图3 编译失败时的运行图

**3.4注意事项**

没有时间处理UI界面，操作较为繁复，原计划采用web前端+后端的UI方案，采用Django服务框架。UI界面将后续实现。

**四、项目架构**

一个经典的编译程序一般包括7个部分：词法分析，语法分析，语义分析及代码生成，代码优化（可省略），代码执行，符号表管理，出错管理。这7个部分之间的关联关系如下图所示：

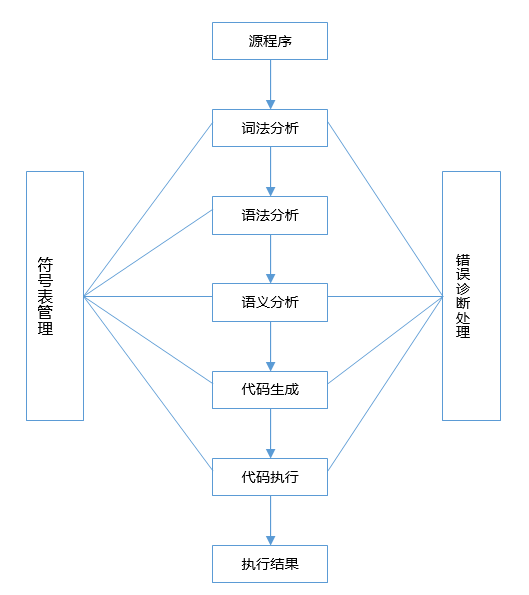


图4 编译程序各部分之间的关联关系

本项目将整个编译过程分为了3个部分：

cifa.py将Pl/0源代码分为一个个token

yufa.py完成语法分析、语义分析和生成Pcode（包括符号表管理和出错管理）工作。

Interpreter.py完成对编译成功代码的pcode的解释工作

整个编译过程可以如下图所示：

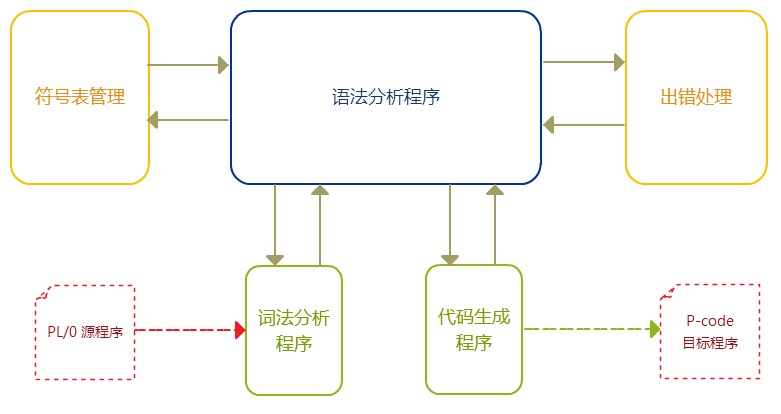


图5 编辑过程流程图

下面，对本工程中的各个部分一一做介绍：

**（1）词法分析**

PL/0编译系统中所有的token的类型为，如下表格：

|  |  |
| --- | --- |
| 关键字 | const、var、procedure、odd、if、then、else、while、do、  call、begin、end、repeat、until、read、write |
| 单字运算符 | +、 -、 =、 \*、 /、 <、 > |
| 双字运算符 | <> 、<= 、>= 、:= |
| 标识符 | x, y, z等 |
| 常数 | 256，128等整形数字 |
| 分界符 | , 、; 、( 、)、. |

表2 PL/0编译系统中所有token的类型

Token类定义如下

class Token(object): # 字符类（存储词法分析结果）  
 def \_\_init\_\_(self, symtype, line, value):  
 self.symtype = symtype # 字符类型（详见cifa.py开头注释）  
 self.line = line # token所在行号  
 self.value = value # 字符值（string）

图6 词法分析程序的状态转换图

**（2）符号表管理**

1. 符号表结构

class Symbol(object): # 符号类（符号表项）  
 def \_\_init\_\_(self, symtype, value, level, address, size, name):  
 self.symtype = symtype # 符号类型 const: 1 var: 2 proc: 3 (int)  
 self.value = value # 符号值 const: value var: -1 proc: Pcode address (int)  
 self. level = level # 层数 level of block (int)  
 self.address = address # 地址 address (int)  
 self.size = size # 大小 (int) 实际上没用到  
 self.name = name # 符号名称 name of the symbol (string)

我们可以看到，每个symbol明显要比token复杂的多，相关变量也复杂的多。其中level和address在运行时起到非常大的作用。

1. 符号表管理

|  |  |
| --- | --- |
| enter\_const | 新增常量表项 |
| enter\_var | 新增变量表项 |
| enter\_proc | 新增过程表项 |
| exist\_now | 当前层存在 |
| exist\_pre | 之前定义过 |
| get\_symbol | 获取symbol对象 |
| get\_level\_proc | 得到该层起始过程 |

表3 各函数名功能表示图

**（3）语法分析和语义分析**

使用递归下降子程序法，对每一个PL/0中的语法成分都进行了分析，并单独编写为一个过程。此处不再赘述。

**（4） Pcode生成：**

P-code 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。

|  |  |  |
| --- | --- | --- |
| F | L | A |

指令格式:

F ：操作码

L ：层次差 （标识符引用层减去定义层）

A ：不同的指令含义不同

在本项目中用到的Pcode如下：

|  |  |
| --- | --- |
| LIT | 取常量加到栈顶 |
| INT | 申请栈空间 |
| LOD | 加载变量值到栈顶 |
| RED | 读操作 |
| WRT | 写操作 |
| OPR | 具体操作取决于A |

表5 P-code 指令的含义

地址回填

对于可能出现的跳转语句，需要采用地址回填。

if-then语句的目标代码生成模式：

|  |  |
| --- | --- |
| if <condition> then <statement> | |
|  | |
|  | <condition> |
|  | JPC addr1 |
|  | <statement> |
| addr1: |  |

If-then-else语句的目标代码生成模式：

|  |  |
| --- | --- |
| if <condition> then <statement>[else] | |
|  | |
|  | <condition> |
|  | JPC addr1 |
|  | <statement> |
|  | JMP addr2 |
| addr1: | [else] |
|  | <statement> |
| addr2 |  |

while-do语句的目标代码生成模式：

|  |  |
| --- | --- |
| while <condition> do <statement> | |
| addr2: | <condition> |
|  | JPC addr3 |
|  | <statement> |
|  | JPC addr2 |
| addr3: |  |

repeat-until语句的目标代码生成模式：

|  |  |
| --- | --- |
| repeat <statement> until <condition> | |
| addr4: | <statement> |
|  | <condition> |
|  | JPC addr4 |

**(5) 出错管理：**

本项目将错误分为26类，其中包含了语法错误和语义错误，详见main.py的show\_log()函数内定义。