# 目录

[词法详细设计文档 1](#_Toc2640)

[1. 引言 1](#_Toc19930)

[2. 词法分析器概述 1](#_Toc18537)

[3. 详细设计 1](#_Toc20008)

[3.1 预处理 1](#_Toc2147)

[3.2 词法单位 1](#_Toc9884)

[3.3 数据结构 3](#_Toc12902)

[3.4. 错误处理 4](#_Toc11223)

[3.5 函数和过程 6](#_Toc14475)

[4. 使用说明 8](#_Toc30980)

[4.1 命令行参数 8](#_Toc20167)

[4.2 输出结果 9](#_Toc26404)

# 词法详细设计文档

## 1. 引言

本报告旨在详细描述一个基于Python和ply.lex库实现的词法分析器的设计和实现。该词法分析器专门针对Pascal语言进行词法分析，能够将源代码文本分解成一系列的标记（Token），并提供错误处理机制。

## 2. 词法分析器概述

词法分析器是编译器或解释器前端的重要组成部分，负责将源代码文本流分解成一个个的Token，同时识别出源代码中的注释和常量等结构。本词法分析器的设计目标是准确、高效地完成这一任务，并能够处理常见的词法错误。

## 3. 详细设计

### 3.1 预处理

PASCAL-S大小写不敏感，因此需要将PASCAL-S源程序中的所有大写字母转化为小写字母。并且pas文件读取时空格有时会乱码，故也需要转化：

self.data=''.join(dataline).replace('\xa0',' ').lower()#转空格、小写

### 3.2 词法单位

本词法分析器识别以下主要词法单位：

|  |  |  |  |
| --- | --- | --- | --- |
| **描述** | **正则表达式** | **属性** | **记号** |
| **标识符** | **[a-z][a-z0-9]\*** | **该标识符本身** | **ID** |
| **整数** | **r'\d+'** | **该整数本身 (int)** | **DIGITS** |
| **浮点数** | **r'\d+\.\d+'** | **该浮点数本身(float)** | **NUM** |
| **字符常量** | **r'\'[^\']\*\'\*'** | **该字符常量本身（不包含两侧的单引号）** | **LETTERS** |
| **关系运算符** | **>=** | **>=** | **RELOP** |
| **>** | **>** |
| **<=** | **<=** |
| **<>** | **<>** |
| **<** | **<** |
| **关系运算符：相等**  **常量初始化：赋值** | **=** | **=** | **EQUAL** |
| **算术运算符：加法** | **+** | **+** | **ADDOP** |
| **逻辑运算符：或** | **or** | **or** |
| **算术运算符：减法** | **-** | **-** |
| **算术运算符：乘法** | **\*** | **\*** | **MULOP** |
| **算术运算符：除法** | **/** | **/** |
| **算术运算符：整除** | **div** | **div** |
| **算术运算符：取余** | **mod** | **mod** |
| **逻辑运算符：且** | **and** | **and** |
| **逻辑运算符：非** | **not** | **not** | **NOT** |
| **赋值符号** | **r‘:=’** | **:=** | **ASSIGNOP** |
| **范围连接符** | **r'\.\.'** | **..** | **DOT** |
| **界符** | **r'\('** | **(** | **LPAREN** |
| **r‘\)’** | **)** | **RPAREN** |
| **r‘\[’** | **[** | **LBRACKET** |
| **r‘\]’** | **]** | **RBARCKET** |
| **r‘:’** | **:** | **COLON** |
| **r‘,’** | **,** | **COM** |
| **r‘;’** | **;** | **SEMICOLON** |
| **r‘\.’** | **.** | **POINT** |
| **关键字** | **program** | **program** | **PROGRAM** |
| **关键字** | **const** | **const** | **CONST** |
| **关键字** | **var** | **var** | **VAR** |
| **关键字** | **array** | **array** | **ARRAY** |
| **关键字** | **of** | **of** | **OF** |
| **关键字** | **procedure** | **procedure** | **PROCEDURE** |
| **关键字** | **function** | **function** | **FUNCTION** |
| **关键字** | **begin** | **begin** | **BEGIN** |
| **关键字** | **end** | **end** | **END** |
| **关键字** | **if** | **if** | **IF** |
| **关键字** | **then** | **then** | **THEN** |
| **关键字** | **for** | **for** | **FOR** |
| **关键字** | **to** | **to** | **TO** |
| **关键字** | **do** | **do** | **DO** |
| **关键字** | **else** | **else** | **ELSE** |
| **关键字** | **repeat** | **repeat** | **REPEAT** |
| **关键字** | **until** | **until** | **UNTIL** |
| **关键字** | **while** | **while** | **WHILE** |
| **关键字** | **integer** | **integer** | **INTEGER** |
| **关键字** | **real** | **real** | **REAL** |
| **关键字** | **char** | **char** | **CHAR** |
| **关键字** | **boolean** | **boolean** | **BOOLEAN** |

### 3.3 数据结构

#### Lexer类

为了方便其他模块调用此LEX模块，我将整体代码集成为类

class Lexer:

    error=[] #错误列表 [dict]

    data=None #输入数据 数据结构为str字符串

COMMENT={} #注释集合字典 （行号，列号）："注释内容"

def build(self,filename) #初始化lex

def find\_column(self,data,token) #寻找token列号

def debug(self,debug) #调试，输出token流和错误列表

各成员或方法用处见注释,需要补充的如下：

#### error

数据结构为列表中嵌套二维字典。具体示例：

#添加方法：

self.error.append({  # 不在已有错误中，则为词法分析中的非法字符错误

                   "code": "A-03",

                  "info": {

                          "line": t.lineno,

                          "value": [t.value.split('\n')[0]],

                          "column": self.find\_column(self.data,t)

                        }

               })

如注释右花括号缺失错误加入后：

error：

[{'code': 'A-09',

'info': { 'line': 4,

'value': ['Comment\_Eof'],

'column': 8}}]

#### COMMENT

数据结构为字典，键为元组，值为字符串。具体示例：

#添加方法：

self.COMMENT[(t.lineno,self.find\_column(self.data,t))]=t.value[1:-1]

如：

{ (2, 1): ' gfylg\nucflgk',

(4, 8): 'lixsaniufcbasiu' }

#### lex内置数据结构

**1. lexer.lexpos**

这是一个表示当前分析点的位置的整型值。如果你修改这个值的话，这会改变下一个token()的调用行为。在标记的规则方法里面，这个值表示紧跟匹配字串后面的第一个字符的位置，如果这个值在规则中修改，下一个返回的标记将从新的位置开始匹配

**2. lexer.lineno**

表示当前行号。PLY只是声明这个属性的存在，却永远不更新这个值。

**3. lexer.lexdata**

当前lexer的输入字串，这个字符串就是input()方法的输入字串

### 3.4. 错误处理

#### 3.4.1. 错误种类及其对策

1. **行长度超过限制A-01** 本编译器设置为100；遇到该错误时报错，且词法分析器立即停止运行
2. **标识符长度超过限制A-02** 本编译器设置为20；遇到该错误时报错，并对标识符进行截断处理，只取前20个字符，这可能会造成后续语义分析中出现重定义的错误
3. **非法字符A-03** 遇到该错误时报错并跳过该非法字符
4. **标识符以数字开头A-04** 遇到该错误时报错，返回去除首字符数字后的标识符
5. **读取字符常量时遇到文件尾A-05** 遇到该错误时报错，词法分析器跳过一行，返回首字符作为字符常量
6. **读取的字符常量为空A-06** 遇到该错误时报错，并返回"\0"这一字符常量
7. **读取的字符常量不止一个字符A-07** 遇到该错误时报错，返回首字符作为字符常量
8. **读取字符常量先遇到换行符而不是单引号A-08** 遇到该错误时报错，如果之前没有识别任何字符，则返回”\0”，如果之前识别了字符，则返回第一个字符
9. **读取多行注释时遇到文件尾A-09** 也就是右花括号缺失的情况；遇到该错误时报错，返回左花括号后首行字符串作为注释，这可能会引起后续语法分析中出现语法错误

#### 3.4.2. 错误信息格式

**1. 存储格式**

和[{'code': 错误代码,

'info': { 'line': 行号,

'value': 错误内容, 'column': 列号}}

1. **输出格式**

code line column value

错误编号 行号 列号 错误内容

1. **错误信息**

|  |  |  |
| --- | --- | --- |
| 错误编号 | 描述 | Value |
| A-01 | 行长度超出限制 | Line length too large, exceed 100！ |
| A-02 | 标识符长度超出限制 | [Identifier length too large, exceed 20] : 标识符 |
| A-03 | 非法字符 | [Invalid character!] ：标识符 |
| A-04 | 标识符以数字开头 | [Wrong Identifier]: 标识符 |
| A-05 | 读取字符常量时遇到文件尾 | Unexpected end of file when reading a char constant |
| A-06 | 读取的字符常量为空 | Char constant missing! |
| A-07 | 读取的字符常量不止一个字符 | Too many characters in a char constant! |
| A-08 | 读取字符常量时，先遇到了换行符而不是右单引号 | Right quote missing! |
| A-09 | 读取多行注释时遇到文件尾 | Unexpected end of file when reading a multiple line comment, lacking of a right brace |

### 3.5 函数和过程

#### 3.5.1 Lex正则表达式函数

使用Lex正则表达式方法实现对Token的优先级识别规则与排错，按顺序展示如下：

1. **#通过对参数t(当前token)的操作实现以下功能**
   1. **t.value 为当前token的内容**
2. **def t\_LETTERS(t): #对字符常量的识别与排错 t.value为string**
3. **def t\_NUM(t): #对浮点数的识别 t.value为float**
4. **def t\_ID(t): #对标识符和关键字和整数的识别，对标识符的排错** 
   1. **r'[0-9a-zA-Z\_][a-zA-Z\_0-9]\*' #识别整数和字母的任意组合**
   2. **t.type = dict(reserved, \*\*reserved\_2).get(t.value,'ID') #检查是否为关键字**
   3. **if t.value.isdigit(): #检查是否为整数**
   4. **t.type='DIGITS'**
   5. **t.value=int(t.value)**
   6. **if(t.type=='ID'): #实现对ID的排错**
5. **def t\_DIGITS(t): #对浮点数的识别 t.value为int 放在t\_ID后以便实现对标识符的排错**
6. **def t\_newline(t): #实现对行号的追踪**
7. **def t\_error(t): #对非法字符的处理 报错并跳过**

#### 3.5.2 对注释的处理函数

注释(comment):{}包含的任意内容，由于{}可以嵌套，用正则表达式识别几乎不可能，故我使用Lex库内置的状态栈实现对注释中嵌套{}的识别：

伪代码如下：

1. **#状态栈：**
2. **states = (('comment','exclusive'),)**
3. **#本质上是用栈的方法实现左右括号嵌套识别的**
4. **# Match the first {. Enter comment state.**
5. **def t\_comment(t):**
   1. **r'\{'**
   2. **# 进入comment状态**
6. **# Rules for the comment state**
7. **def t\_comment\_lbrace(t):**
8. **r'\{'**
9. **t.lexer.level +=1 #level+1表示左括号进栈**
10. **def t\_comment\_rbrace(t):**
    1. **r'\}'**
    2. **t.lexer.level -=1 #level-1表示右括号匹配成功，出栈**
    3. **if t.lexer.level == 0: #level=0时表示栈为空，识别成功**
       1. **...保存注释、更新行号、回归原状态...**
11. **def t\_comment\_ID(t):**
    1. **r'[^{}]+'**
    2. **if t.value[-1] != self.data[-1] :**
    3. **Pass #跳过**
    4. **else: #表明右花括号缺失**
    5. **...添加error...**
    6. **flag = t.lexer.lexpos+(t.value.find('\n')-len(t.value))**
    7. **#flag设置为回移指针，跳一行实现转为单行注释的恢复方法**
    8. **...保存单行注释内容，更新行号，返回原状态...**
12. **t\_comment\_ignore = " " #忽视字符**
13. **def t\_comment\_error(t): #跳过**

#### 3.5.3辅助函数

* find\_column(self,data,token)：计算Token在源代码中的列位置。

|  |  |
| --- | --- |
| 参数 | 描述 |
| Class self | 指向自己的类对象 |
| string data | 输入文件流（字符串） |
| Class token | 目标token |

返回值：int

代码：

1. **def find\_column(self,data,token):**
2. **last\_cr = data.rfind('\n',0,token.lexpos) #上一个’\n’的偏移量**
3. **if last\_cr < 0:**
4. **last\_cr = 0**
5. **column = token.lexpos - last\_cr #列号=目标偏移量-‘\n’偏移量**
6. **return column**

* def debug(self)：用于调试和输出Token信息。

|  |  |
| --- | --- |
| 参数 | 描述 |
| Class self | 指向自己的类对象 |

返回值：void

#### 3.5.4 构建和运行

构建词法分析器。

* 函数接口

1. def build(self,filename,\*\*kwargs)
2. 返回值：void

|  |  |
| --- | --- |
| 参数 | 描述 |
| Class self | 指向自己的类对象 |
| String filename | 文件名 |
| \*\*kwargs | 多余参数 |

伪代码：

1. def build(self,filename,\*\*kwargs)：
2. ...关键字reserved,reserved\_2;tokens...
3. ...4.1内所有正则表达式规则与方法...
4. self.lexer = lex.lex( ) #初始化
5. with open(filename,'r',encoding='utf-8') as file:
   1. dataline=file.readlines()
   2. for i in range(0,len(dataline)):
   3. ...判断行长度是否超出限制..
6. self.data=dataline列表中各项拼接而成的字符流
7. self.lexer.input(self.data) #开始词法识别
8. if kwargs['kwargs']==1:
9. self.debug() #输出token流
10. if self.error != []:
    1. ...输出error列表...

## 4. 使用说明

### 4.1 命令行参数

词法分析器通过命令行参数接收输入文件和调试标志：

* 第一个参数为源代码文件的路径。
* 第二个参数为调试标志，用于控制是否输出详细的Token内容。
* 入口：Lexer.build(sys.argv[1],kwargs=sys.argv[2])

### 4.2 输出结果

词法分析器将输出以下信息：

* 分析出的Token及其位置信息。
* 遇到的词法错误及其位置信息。
* 源代码中的注释。