

**编译原理实验报告**

学院: 计算机学院

学号: 17061833

学生姓名： 於文卓

手机号： 17857158661

电子邮箱： 435212619@qq.com

目录

[前言 3](#_Toc11086986)

[词法分析器 3](#_Toc11086987)

[1、目的 3](#_Toc11086988)

[2、过程 3](#_Toc11086989)

[3、实验结果 6](#_Toc11086990)

[语法分析器 6](#_Toc11086991)

[1、目的 6](#_Toc11086992)

[2、过程 7](#_Toc11086993)

[3、实验结果 9](#_Toc11086994)

[语义分析器 10](#_Toc11086995)

[1、目的 10](#_Toc11086996)

[2、过程 11](#_Toc11086997)

[3、实验结果 12](#_Toc11086998)

[遇到的困难 13](#_Toc11086999)

[收获 14](#_Toc11087000)

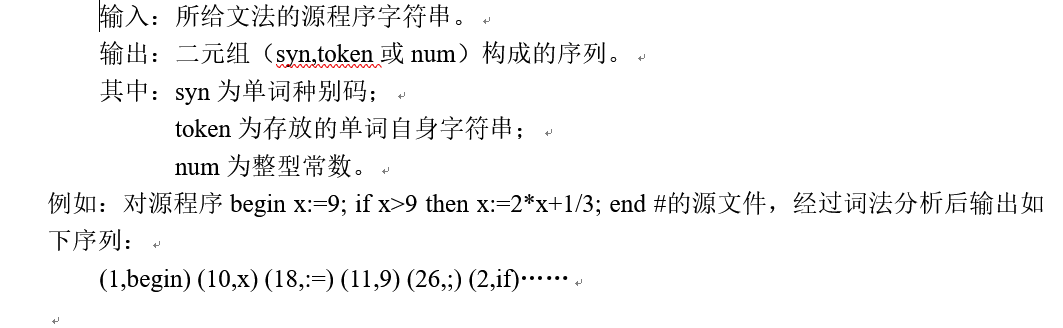
## 前言

编译原理是计算机专业的一项基础的课程，是研究软件是什么，为什么可以运行，以及怎么运行的学科，编译系统的改进将会直接对其上层的应用程序的执行效率，执行原理产生深刻的影响。编译原理的目的是将源语言翻译成目标语言。本次实验将使用python实现词法分析、语法分析和语义分析。

## 词法分析器

### 1、目的

设计、编制并调试一个词法分析程序，加深对词法分析原理的理解。

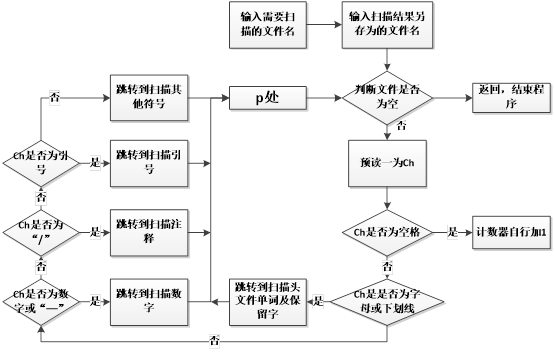


### 2、过程

#### 流程图



从主函数开始，一次读入文件中的字符流，在没有读到字符结尾#符号之前，判断每一个字符。对首字符是字母、数字、操作符和界符分别做处理，如果首字符是字母，接下去判断下一个字符是否是数字，或者是否已经组成关键字。



#### 代码

**import** **re**

dic\_keyword = {

'begin':1,

'if' : 2,

'then':3,

'while':4,

'do':5,

'end':6,

}

dic\_ID = {'ID':10}

dic\_Num = {'Num':11}

dic\_operator = {

'+':13,

'-':14,

'\*':15,

'/':16,

':':17,

':=':18,

'<':20,

'<>':21,

'<=':22,

'>':23,

'>=':24,

'=':25,

';':26,

'(':27,

')':28,

'#':0

}

**def** tokenize(code):

ERROR = r'(?P<ERROR>\,|(\d+\w+))'

keywords = r'(?P<Keyword>(begin){1}|(if){1}|(then){1}|(while){1}|(do){1}|(end){1})'

Operator = r'(?P<Operator>\:=|\:|\+|\-|\\*|\/|<=|>=|<>|>|<|=|;|\(|\)|#)'

ID = r'(?P<ID>[a-zA-Z\_][a-zA-Z\_0-9]\*)'

Num = r'(?P<Num>\d+(\.\d\*)?)'

res = '|'.join([ERROR,keywords,Operator,Num,ID])

patterns = re.compile(res)

**for** one **in** re.finditer(patterns,code):

*# print(one)*

**if** one.lastgroup == 'Keyword':

**print**('(',dic\_keyword[one.group()],',',one.group(),')')

**elif** one.lastgroup == 'Operator':

**print**('(',dic\_operator[one.group()],',',one.group(),')')

**elif** one.lastgroup == 'Num':

**print**('(',dic\_Num['Num'],',',one.group(),')')

**elif** one.lastgroup == 'ID':

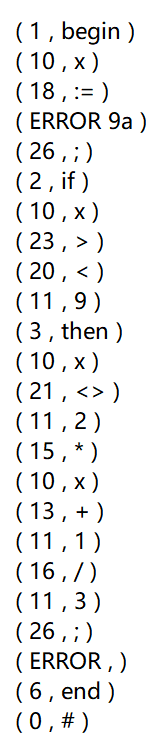
**print**('(',dic\_ID['ID'],',',one.group(),')')

**elif** one.lastgroup == 'ERROR':

**print**('(',"ERROR",one.group(),')')

tokenize('begin x:=9a; if x><9 then x<>2\*x+1/3;, end #')

### 3、实验结果



## 

## 语法分析器

### 1、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。

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：产生式序列，并给出判断串是否合法。

4. 验收测试用例：1）i++i\*\*i； 2） (i+i\*i)+i；3）i+i\*i+。

### 2、过程

#### 构造FIRST集

*#获取first集，传入的参数是某一条产生式*

**def** getFirst(syntax):

**for** j **in** range(len(syntax)-1):

**for** i **in** syntax:

syntaxSplit = i.split('->')

left = syntaxSplit[0]

right = syntaxSplit[1]

**if** right == '#':

FIRST[left].append(right)

**elif** right[0] **in** VT:

FIRST[left].append(right[0])

**else**:

FIRST[left].extend(FIRST[right[0]])

FIRST[left] = list(set(FIRST[left]))



#### 构造FOLLOW集

**def** getFollow(syntax):

*#文法的开始符号*

start = syntax[0].split('->')[0]

FOLLOW[start].append("$")

**for** i **in** range(len(syntax)-1):

**for** i **in** syntax:

syntaxSplit\_1 = i.split('->')

*#左部*

left\_1 = syntaxSplit\_1[0]

**for** j **in** syntax:

syntaxSplit\_2 = j.split('->')

left\_2 = syntaxSplit\_2[0]

right\_2 = syntaxSplit\_2[1]

**if** left\_1 **in** right\_2:

index = right\_2.find(left\_1)

*#如果index可以加一*

**if** index+1 < len(right\_2):

*#index+1的地方的值是终结符*

**if** right\_2[index+1] **in** VT:

*#那FOLLOW集加入终结符*

FOLLOW[left\_1].append(right\_2[index+1])

**else**:

*#如果不是终结符，加他的FIRST集，再分两种情况 有空串 和 没有空串*

FOLLOW[left\_1].extend(FIRST[right\_2[index+1]])

*#如果有空串*

**if** '#' **in** FIRST[right\_2[index+1]]:

FOLLOW[left\_1].remove('#')

FOLLOW[left\_1].extend(FOLLOW[left\_2])

*#去重*

FOLLOW[left\_1] = list(set(FOLLOW[left\_1]))

*#如果index不可以+1 T->FT' 放在T后面也要放在T'后面*

**if** index+1 == len(right\_2):

FOLLOW[left\_1].extend(FOLLOW[left\_2])

FOLLOW[left\_1] = list(set(FOLLOW[left\_1]))



#### 构造SELECT集

**def** getSelect(syntax):

**for** i **in** syntax:

syntaxSplit = i.split('->')

left = syntaxSplit[0]

right = syntaxSplit[1]

**if** right[0] **in** VT:

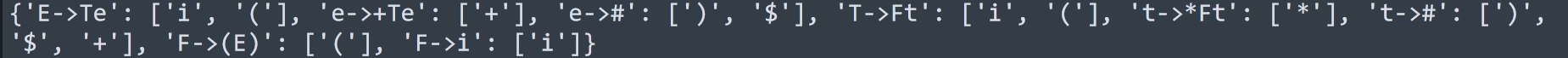
SELECT[i].append(right[0])

**elif** right == '#':

SELECT[i]= FOLLOW[left].copy()

**else**:

SELECT[i]= FIRST[left].copy()



#### 构造预测分析表

使用python的字典来存储预测分析表中的值

**def** preTable(syntax):

**for** i **in** syntax:

syntaxSplit = i.split('->')

left = syntaxSplit[0]

right = syntaxSplit[1]

tmp = {}

**for** j **in** SELECT[i]:

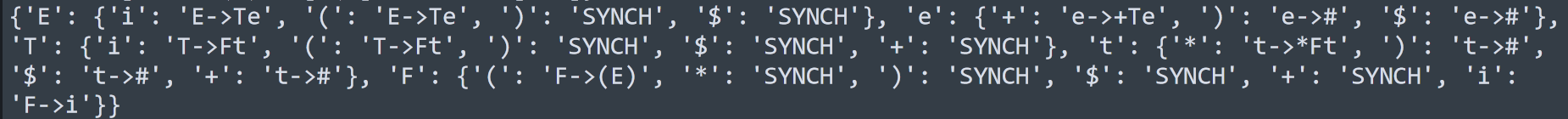
tmp[j] = i

**for** j **in** FOLLOW[left]:

**if** tmp.get(j) == None:

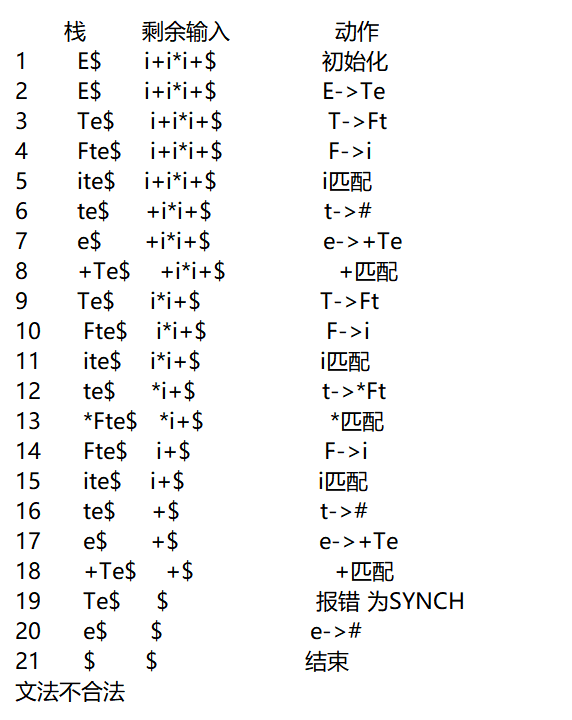
tmp[j] = 'SYNCH'

TABLE[left].update(tmp)



### 3、实验结果

输入文法：'i+i\*i+$'



## 语义分析器

### 1、目的

1. 了解语义分析的主要方法和功能，能判断语法制导定义类型，并根据不同类型采用不同的语义分析方法。

2. 掌握利用语义规则进行符号表信息登录的功能。

### 2、过程

#### Enter函数

**def** enter(table, name, typename, offset):

item = stitem(name, typename, offset)

table.push(item)

#### Mktable函数

**def** mktable():

table = tbl()

**return** table

#### Addwidth 函数

**def** addwidth(table, width):

table.width = width

#### Enterproc 函数

**def** enterproc(table, name, newtable):

table.addTable(name,newtable)

**return** table

通过产生式规则，自底向上，手动构建语法分析树后，得到最左规约的产生式。根据产生式的顺序，执行对应的语义规则

如对于测试句子：id1 : real ; id2 :↑integer; id3:integer

**def** testOne():

T1 = T(name="T1",state=1)

id1 = ID(name="id1",state=1)

t1 = mktable()

tblptr.append(t1)

offset.append(0)

T1.type\_ = 'real'

T1.width = 8

enter(tblptr[-1],id1.name,T1.type\_,offset[-1])

offset[-1] = offset[-1] + T1.width

*# print(offset)*

T3 = T(name="T3",state=3)

T3.type\_ = "integer"

T3.width = 4

T2 = T(name="T2",state=2)

T2.type\_="ptr "+str(T3.type\_)

T2.width = 4

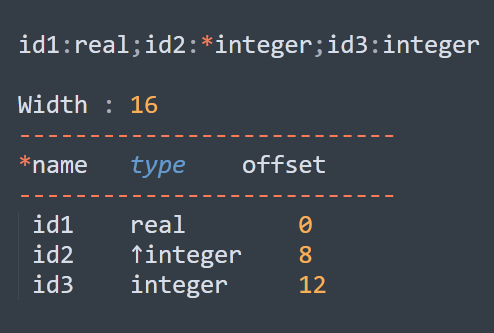
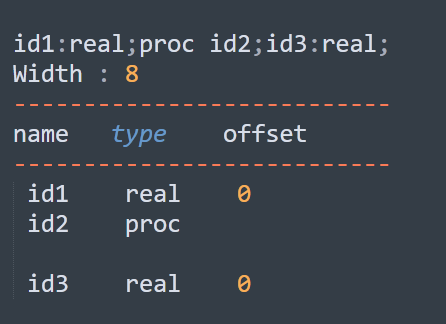
id2 = ID("id2",2)

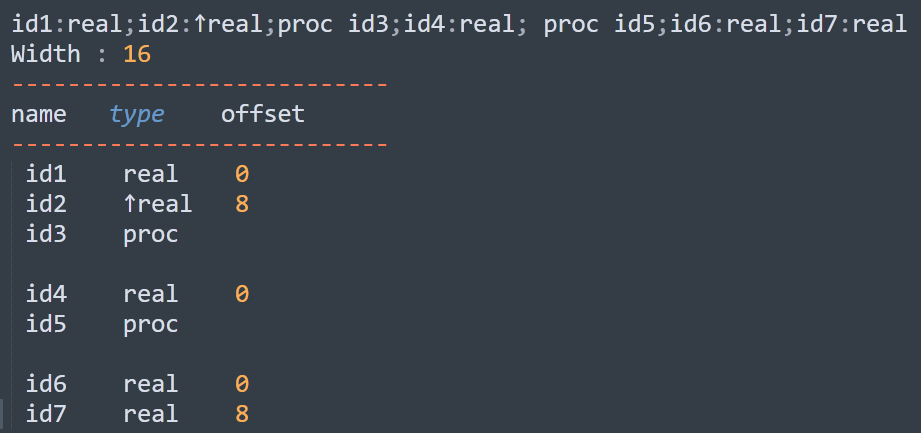
enter(tblptr[-1],id2.name,T2.type\_,offset[-1])

offset[-1] = offset[-1] + T2.width

**return** t1.show()

### 3、实验结果



## 遇到的困难

#### 词法分析

词法分析其实是一个手动实现正则表达式的过程，实现过程中需要对输入字符的不同情况进行判断，很容易因为考虑不周全，逻辑不严谨导致出现漏掉条件的情况。实现过程使用了很多if else 代码在重新阅读的时候看起来容易把自己搞晕。

#### 语法分析

语法分析器我是自己通过实现first集，follow集，select集来实现预测分析表。所以工作量还是比较大的。但是每一步实现只要按部就班做其实还是容易的，实现过程中除了耗时间，还挺顺利。

#### 语义分析

本来是想通过LR（0）分析器，自动构建自底向上的分析树，实现边分析边翻译的，奈何时间、能力不够，没法完成，最后就通过自己手动构建分析树的方法，来确定执行语义规则的顺序。这样就简单了很多

## 收获

编译原理对于计算机专业来说是一门非常重要的课，曾在网上看到有大神自己写了一个编译器，创造了一门自己的语言。在没学这门课时，想着学了这门课我自己是不是也能发明一个自己的语言。而学习这门课的时候，发现里面的概念抽象且数量庞大，一时间失去了学下去的方向和动力。还好有实践课，让我可以从编译器几个小的步骤开始，如实现简单的词法分析、语法分析、语义分析，通过动手的方式更好的去理解这些概念。通过这种以小见大的方式，我也对编译原理有了更进一步的理解和认识