

电子科技大学

实验报告

学生姓名：岳子豪 学号：2018051404015 指导教师：陈昆

实验地点：科 A504 实验时间：2021.5.15

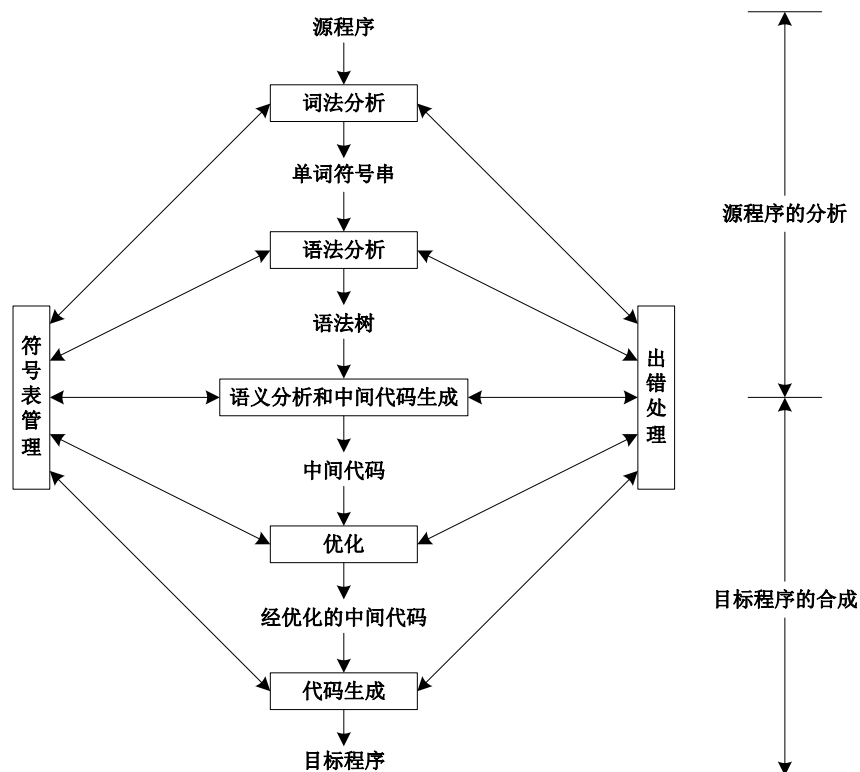
一、实验室名称：计算机学院软件工程实验室

二、实验项目名称：词法分析器的设计与实现

三、实验学时：4 学时

四、实验原理

编译程序要求对高级语言编写的源程序进行分析和合成，生成目标程序。词法分析是对源程序进行的首次分析，实现词法分析的程序为词法分析程序。

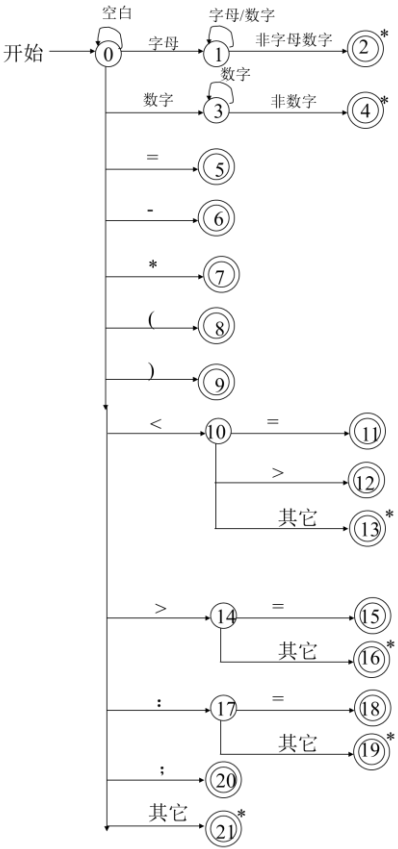


词法分析负责从左到右逐个地扫描源程序字符串，按照词法规则识别出单词符号作为输出，对识别过程中发现的词法错误，输出相关信息，其输入为源程序代码，输出为单词符号串，即由单词符号串和标号构成的二元组，用于下一阶段的语法分析。

本实验中，为实现求 $n!$ 的极小语言的词法分析程序，所用到的单词符号和其对应的标号如下表所示。

单词符号	种别	单词符号	种别	单词符号	种别
begin	1	end	2	integer	3
if	4	then	5	else	6
function	7	read	8	write	9
标识符	10	常数	11	=	12
<>	13	<=	14	<	15
>=	16	>	17	-	18
*	19	:=	20	(21
)	22	;	23		

对于词法分析而言，状态转换图是有限有向图，是设计词法分析器的有效工具。根据状态转换图设计程序的执行逻辑，能准确有效地完成词法分析任务。本实验中所用到的词法的状态转换图如下。



五、实验目的

通过设计词法分析器的实验，使同学们了解和掌握词法分析程序设计的原理及相应的程序设计方法，同时提高编程能力。

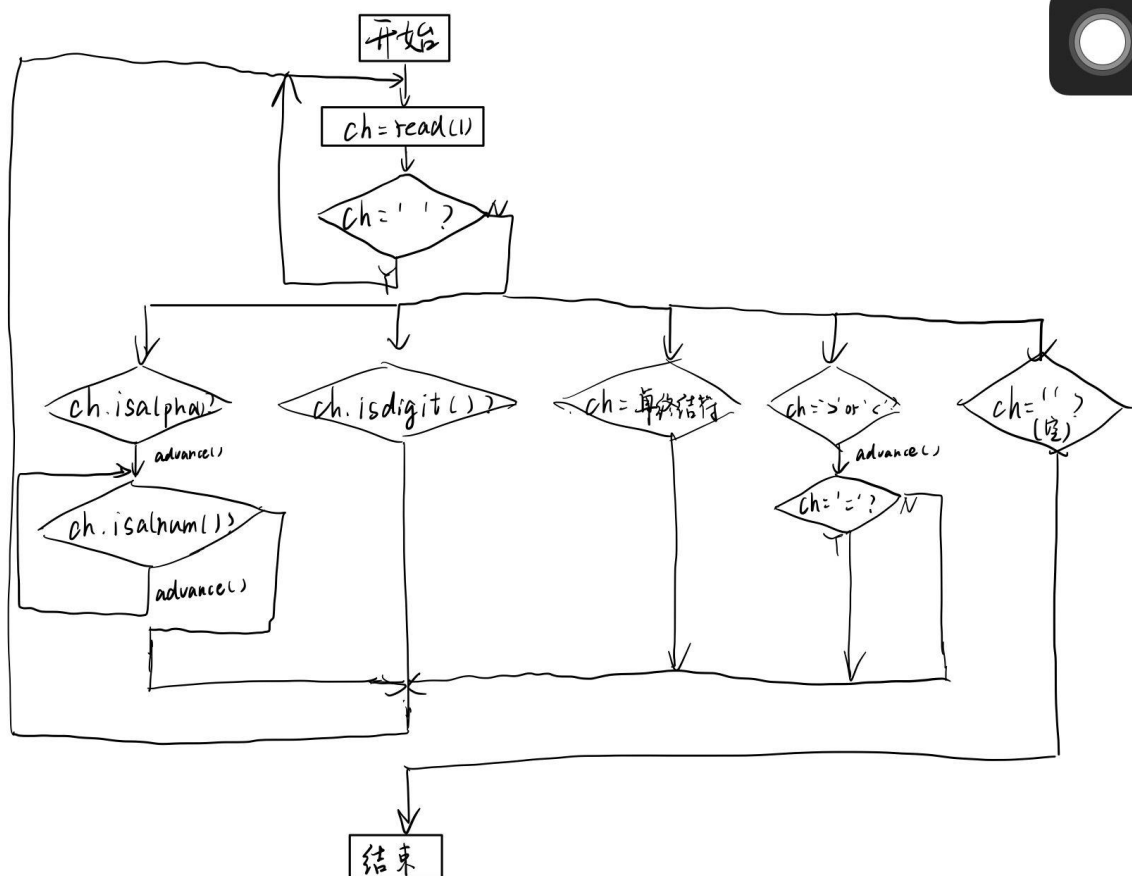
六、实验内容

实现求 $n!$ 的极小语言的词法分析程序，返回二元式作为输出。

七、实验器材（设备、元器件）

1. 操作系统：Windows 10
2. 开发工具：PyCharm 2020.1 x64
3. 笔记本电脑

八、实验步骤（此步骤给出算法流程图即可）



由于过程细节繁琐，故只给出整体框架，具体实现详见如下分析及源代码。

定义一个全局列表 `symbol_list` 用于存储所有非终结符。

```
symbol_list=[0, 'begin', 'end', 'integer', 'if', 'then', 'else', 'function', 'read',  
'write', 10, 11, '=', '<>', '<=', '<', '>=', '>', '-', '*', ':=', '(', ')', ';']
```

主函数使用一个 `while` 循环作为整体逻辑实现，首先打开源代码文件 `sample.pas`、`sample.dyd`、`sample.err`，然后开始循环，每次读取一个字符，用于接下来的分析。`error_type` 用于记录错误类型，`temp` 字符串作为临时变量用于存储读取到的非终结符，初始化为空。`idx` 作为行内指针，`line` 作为行号。

```
if __name__ == '__main__':  
  
    input=open('sample.pas','rb')  
    output=open('sample.dyd','w')  
    error=open('sample.err','w')  
  
    idx=0  
    line=1  
  
    while True:  
  
        error_type=0  
        ch=input.read(1)  
        idx+=1  
        temp=b''
```

接下来进行词法分析。

分析的过程中，始终跳过空格，若遇到换号符号，则跳过，并向输出文件添加 `EOLN`。若遇到空字符，则说明文件已经结束，退出循环，并向输出文件中添加 `EOF`。

```
if ch==b' ':           # 空格  
    continue  
  
elif ch==b'\n' or ch==b'\r':   # 换行  
    ch=input.read(1)  
    temp=b'\n'  
    line+=1  
    idx=0  
    output.write('{} {}{}\n'.format('EOLN'.rjust(16, ' '), 24))
```

```

        continue

elif ch==b'':          # 文件结束
    temp=b''
    output.write('{} {}\\n'.format('EOF'.rjust(16, ' '), 25))
    break

```

若接收到的字符是字母，则进入循环，准备接受一个字符串终结符。从第二个字符开始，如果是字母或数字，则继续，否则退出循环，指针会退一格。

```

elif ch.isalpha():    # 标识符
    while ch.isalnum():
        temp+=ch
        idx+=1
        ch=input.read(1)
        if len(temp)>16:
            error_type=3
            continue
    if ch != b'':
        input.seek(-1, 1)
        idx-=1

```

若接收到的是数字，则准备接受一个长度大于等于 1 的整数。

```

elif ch.isdigit():   # 常数
    while ch.isdigit():
        temp+=ch
        idx+=1
        ch=input.read(1)
        input.seek(-1,1)
        idx-=1

```

若接收到的是单字符终结符，则直接根据全局列表中它们的位置映射，可以使程序更加简洁。

```

elif ch in [b'-' , b'*' , b'(' , b')' , b';']:
    temp=ch

```

如果是双字符终结符，如 ‘:=’，则使用嵌套的 if-else 进行判断。同时，需要记录可能出现的错误的类型。以 ‘:=’ 为例，代码如下：

```

elif ch==b':':
    ch=input.read(1)
    idx+=1
    if ch==b'=':
        temp=b':='

```

```

else:
    if ch!=b'':
        input.seek(-1,1)
        idx-=1
        error_type=1

```

完成一次循环后，得到了一个完整的 temp，然后根据 temp 判断是否为保留字或标识符，写入输出文件。

```

if temp in symbol_list:
    symbol_index=symbol_list.index(temp)
    output.write('{} {} \n'.format(temp.rjust(16,' '),symbol_index))
else:
    output.write('{} {} \n'.format(temp.rjust(16,' '),10))

```

根据分析所记录的错误类型，输出报错结果。
完成上述过程后，保存并关闭文件。

九、 实验数据及结果分析

正确代码示例：

1	begin	1	begin 1	25	then 5	49) 22
2	integer k;	2	EOLN 24	26	F 10	50	; 23
3	integer function F(n);	3	integer 3	27	:= 20	51	EOLN 24
4	begin	4	k 10	28	1 11	52	k 10
5	integer n;	5	; 23	29	; 23	53	:= 20
6	if n<=0 then F:=1;	6	EOLN 24	30	EOLN 24	54	F 10
7	else F:=n*F(n-1);	7	integer 3	31	else 6	55	(21
8	end	8	function 7	32	F 10	56	(21
9	read(m);	9	F 10	33	:= 20	57	m 10
10	k:=F(m);	10	(21	34	n 10	58) 22
11	write(k);	11	n 10	35	* 19	59	; 23
12	end	12) 22	36	F 10	60	EOLN 24
		13	; 23	37	(21	61	write 9
		14	EOLN 24	38	n 10	62	(21
		15	begin 1	39	- 18	63	k 10
		16	EOLN 24	40	1 11	64) 22
		17	integer 3	41) 22	65	; 23
		18	n 10	42	; 23	66	EOLN 24
		19	; 23	43	EOLN 24	67	end 2
		20	EOLN 24	44	end 2	68	EOF 25
		21	if 4	45	EOLN 24	69	
		22	n 10	46	read 8		
		23	>= 16	47	(21		
		24	0 11	48	m 10		

错误代码示例：

sample.pas	sample.err
1 begin	1 LINE:2 标识符 'k1234567890123456' 长度溢出
2 integer k1234567890123456;	2 LINE:5 非法字符 '#'
3 integer function F(n);	3 LINE:6 冒号不匹配
4 begin	
5 integer n#;	
6 if n<=0 then F:1;	
7 else F:=n*F(n-1);	
8 end	
9 read(m);	
10 k:=F(m);	
11 write(k);	
12 end	

可以对源程序进行词法分析，如果有错给出出错信息和所在行数，如果无错则生成二元式文件。

十、实验结论

本实验程序较好地完成了词法分析程序的设计与实现，能够对所给文法的程序进行词法分析，在没有词法错误的时候生成相应的二元式文件。该实验程序可一次性给出源程序中的词法错误。

十一、总结及心得体会

利用 Python 实现了实现求 $n!$ 的极小语言的词法分析程序，能正确有效地运行，分析 pas 源代码并生成 dyd 二元组列表，遇到错误时能识别并展示。通过本次实验，我对词法分析器有了更加深刻的认识，对编译器编译源代码生成目标代码的第一个关键步骤有了更深入的掌握和更透彻的理解。同时，通过自己编写代码实现该词法分析器，提高了自己的逻辑分析能力和编码能力。

十二、对本实验过程及方法、手段的改进建议

程序设计合理，代码可进一步优化。

报告评分：

指导教师签字：

本实验参考源代码见报告结尾。

电子科技大学

实验报告

学生姓名：岳子豪 学号：2018051404015 指导教师：陈昆

实验地点：科 A504 实验时间：2021.5.15

一、实验室名称：计算机学院软件工程实验室

二、实验项目名称：递归下降分析器的设计与实现

三、实验学时：12 学时

四、实验原理

语法分析是对源程序经过词法分析后转换成的单词流按方法规则进行判断，对能构成正确句子的单词流，给出相应的语法树；对不能构成正确句子的单词流判断其语法错误并做出相应处理。

语法分析方法有自上而下和自下而上的分析方法。在不含左递归的文法 G 中，如果对每一个非终结符的所有候选式的第一个终结符都是两两不相交的（即无公共左因子），则可能构造出一个不带回溯的自上而下的分析程序，这个分析程序由一组递归过程组成，每个过程对应文法的一个非终结符。这样的分析程序称为递归下降分析程序。

语法分析所用文法为消除左递归后的文法，如下。

```
<程序>->begin<说明语句表><执行语句表>end
<说明语句表>-><说明语句>B1
B1->;<说明语句>B1|空
<说明语句>-><变量说明>|<函数说明>
<变量说明>->integer<变量>;
<变量>-><标识符>
<函数说明>->integer function<标识符>(<参数>);<函数体>
参数-><变量>
<函数体>->begin<说明语句表>;<执行语句表>end
<执行语句表>-><执行语句>L1
```



```
L1->;<执行语句>L1|空
<执行语句>-><读语句>|<写语句>|<赋值语句>|<条件语句>
<读语句>->read(<变量>)
<写语句>-write(<变量>)
<赋值语句>-><变量>:=<算术表达式>
<算术表达式>-><项>Q1
Q1->-<项>Q1|空
<项>-><因子>R1
R1->*<因子>R1|空
<因子>-><变量>|<常数>|<函数调用>
<条件语句>->if<条件表达式>then<执行语句>else<执行语句>
<条件表达式>-><算术表达式><关系运算符><算术表达式>
<关系运算符>-><|<=>|>=>|<>
```

五、实验目的

通过设计递归下降分析器的设计与实现实验，使同学们掌握自上而下的递归分析法的语法分析原理和程序设计方法。

六、实验内容

根据给定的方法，编写相应的递归下降的语法分析程序，实现对词法分析后的单词序列的语法检查和程序结构的分析，生成相应的变量名表和过程名表，并将编译中语法检查出来的错误写入相应的文件。

语法错分类:

- (1) 缺少符号错
- (2) 符号匹配错
- (3) 符号无定义或重复定义。

七、实验器材（设备、元器件）

1. 操作系统：Windows 10
2. 开发工具：PyCharm 2020.1 x64
3. 笔记本电脑

八、实验步骤

1. 全局变量说明

定义全局指针 idx，行号 line，vector_list 用于存储 sample.dyd 中的二元组，

pro_list 为函数名列表，pvar_list 为函数参数列表，var_list 为变量列表，其中 var_list[0]为主函数变量列表，var_list[1]为子函数变量列表。所有列表全部初始为空，全局指针初始化为 0，行号初始化为 1。

```
global idx
global line
global vector_list
global pro_list
global pvar_list
global var_list

idx=0
line=1
vector_list=[]
pro_list=[]
pvar_list=[]
var_list=[[],[]]
```

2. advance()函数

advance()是语法分析器中最常用的函数，表示当前符号已扫描，指针向前移动一位。当遇到换行时，自动跳过。

```
def advance():
    global idx
    global line
    idx+=1
    if get_tag(idx)==24:
        line+=1
        idx+=1
```

3. 获取当前符号的标号和内容

定义 get_tag()和 get_symbol()用于通过 idx 获取当前指针所指符号的标号或内容。

```
def get_tag(idx):
    global vector_list
    vector=vector_list[idx]
    symbol_tag=vector.split()
    return int(symbol_tag[1].decode())

def get_symbol(idx):
    global vector_list
    vector=vector_list[idx]
    symbol_tag=vector.split()
    return symbol_tag[0].decode()
```

4. 开始()

开始函数表示语法分析的开始，是最顶层的函数。在本函数中，将判断程序的 `begin` 和 `end` 是否正确，并在中间调用说明语句表()和执行语句表()，递归下降分析。(Python 支持中文)

```
def 开始():
    global idx
    global line
    if get_tag(idx)==1:
        advance()
        说明语句表()
        执行语句表(0)
        # print(idx)
    if get_tag(idx)==2:
        return
    else:
        print('开始 LINE{}: expecting \'end\'' .format(line))
        advance()
    else:
        print('开始 LINE{}: missing \'begin\'' .format(line))
        advance()
```

5. 说明语句表()

说明语句表包含变量说明和函数说明，递归下降分别对变量和函数的说明语句进行语法分析。

```
def 说明语句表():
    变量说明(0)
    函数说明()
```

6. 变量说明()

变量说明用于分析定义变量的语句，通过嵌套 `if-else`，在不同的层次对单词符号串进行规约，并定位错误，即时输出。

```
def 变量说明(flag):
    global idx
    global line
    if get_tag(idx)==3:
        advance()
        symbol=get_symbol(idx)
        if get_tag(idx)==10:
            定义了变量(symbol,flag)
            advance()
            if get_tag(idx)==23:
                advance()
```

```

        变量说明(flag)
    else:
        print('变量说明 LINE{}: missing \';\'' .format(line-1))
    elif get_tag(idx)==7:
        idx-=1
        return
    else:
        print('变量说明 LINE{}: invalid variable name
\'{}\'' .format(line,symbol))
        advance()
        if get_tag(idx)==23:
            变量说明(flag)
        else:
            print('变量说明 LINE{}: missing \';\'' .format(line-1))
    else:
        return

```

7. 函数说明()

函数说明包括函数声明和函数过程，其中函数声明用于对函数声明语句进行规约，而函数过程用于对函数体中的执行语句进行规约。

```

def 函数说明():
    函数声明()
    函数过程()

```

8. 函数声明()

函数声明由函数说明调用，用于分析函数声明语句。类似于变量说明，但语法更加复杂，分析时需注意将函数名和参数名存入对应的全局列表。

```

def 函数声明():
    global idx
    global line
    if get_tag(idx)==3:
        advance()
    if get_tag(idx)==7:
        advance()
    if get_tag(idx)==10:
        pro_name=get_symbol(idx)
        advance()
    if get_tag(idx)==21:
        advance()
    if get_tag(idx)==10:
        var_name=get_symbol(idx)
        定义了函数(pro_name,var_name)
        advance()

```

```

        if get_tag(idx)==22:
            advance()
        if get_tag(idx)==23:
            advance()
        else:
            print('函数声明 LINE{}: missing \'';\''.format(line))
        else:
            print('函数声明 LINE{}: expecting \'';\''.format(line))
    elif get_tag(idx)!=22:
        print('函数声明 LINE{}: invalid parameter
\'{}\''.format(line,get_symbol(idx)))
        advance()
    else:
        var_name='null'
        advance()

    else:
        print('函数声明 LINE{}: Syntax error.')
        advance()

    else:
        print('函数声明 LINE{}: invalid function name
\'{}\''.format(line,get_symbol(idx)))
        advance()

    else:
        print('函数声明 LINE{}: Syntax error.'.format(line))
        advance()

else:
    return

```

9. 函数过程()

函数过程中，需对函数体的 begin-end 结构进行判断，并在其中调用变量说明()和执行语句表()对函数体内的执行语句进行分析。

```

def 函数过程():
    global idx
    global line
    if get_tag(idx)==1:
        advance()
        变量说明(1)
        执行语句表(1)
    if get_tag(idx)!=2:
        print('函数过程 LINE{}: expecting \'end\''.format(line))
    else:
        advance()

    return

```

10. 执行语句表(flag)

执行语句表包含执行语句，考虑到程序的执行逻辑，递归将放在执行语句的子函数中进行，在此仅直接调用执行语句(flag)子函数，其中传递的参数 flag 用于调用时通过不同的 flag 值区分主函数执行语句和函数内执行语句。

```
def 执行语句表(flag):  
    执行语句(flag)
```

11. 执行语句(flag)

执行语句包含四类：读语句、赋值语句、写语句和条件语句，先后调用四个子函数，通过递归下降对所有执行语句进行规约。

```
def 执行语句(flag):  
    读语句()  
    赋值语句(flag)  
    写语句()  
    条件语句()
```

12. 读语句()

读语句用于对 read()语句进行规约。在分析时，注意同时定义 read 中的变量并将其添加至变量列表。

```
def 读语句():  
    global idx  
    if get_tag(idx)==8:  
        advance()  
    if get_tag(idx)==21:  
        advance()  
    if get_tag(idx)==10:  
        var_name=get_symbol(idx)  
        定义了变量(var_name, 0)  
        定义了变量(var_name, 1)  
        advance()  
    if get_tag(idx)==22:  
        advance()  
    if get_tag(idx)==23:  
        advance()  
        读语句()  
    else:  
        print('读语句 LINE{}: missing \';\''.format(line-1))  
    else:  
        print('读语句 LINE{}: expecting \')\''.format(line))  
    else:  
        print('读语句 LINE{}: invalid variable name  
'{}\''.format(line,get_symbol(idx)))
```

```

    else:
        print('读语句 LINE{}: expecting \'(\'''.format(line))
else:
    return

```

13. 赋值语句(flag)

赋值语句用于对形如 $xx:=xx+xx*xx$ 的表达式进行规约，本函数只考虑表达式满足 $xx:=xx$ ，具体的算术表达式通过调用子函数递归下降分析。

```

def 赋值语句(flag):
    global idx
    global line
    global pvar_list
    if get_tag(idx)==10:
        if flag==0:
            if get_symbol(idx) in var_list[0]:
                advance()
                if get_tag(idx)==20:
                    advance()
                    算术表达式()
                if get_tag(idx)==23:
                    advance()
                    赋值语句(flag)
            else:
                print('赋值语句 LINE{}: missing \';\'''.format(line))
        else:
            print('赋值语句 LINE{}: Syntax error.'.format(line))
    else:
        print('赋值语句 LINE{}: undefined variable
\'{}\'''.format(line,get_symbol(idx)))
    else:
        if get_symbol(idx) in pro_list or get_symbol(idx) in pvar_list:
            advance()
            if get_tag(idx)==20:
                advance()
                算术表达式()
            if get_tag(idx)==23:
                advance()
                赋值语句(flag)
            else:
                print('赋值语句 LINE{}: missing \';\'''.format(line))
        else:
            print('赋值语句 LINE{}: Syntax error.'.format(line))
    else:

```

```

        print('赋值语句 LINE{}: undefined variable
\{ }\'.format(line,get_symbol(idx)))
    else:
        return

```

14. 写语句()

写语句用于对 write()语句进行规约。

```

def 写语句():
    global idx
    global line
    if get_tag(idx)==9:
        advance()
    if get_tag(idx)==21:
        advance()
    if get_tag(idx)==10:
        if get_symbol(idx) in var_list[0]:
            advance()
            if get_tag(idx)==22:
                advance()
            if get_tag(idx)==23:
                advance()
                写语句()
            else:
                print('写语句 LINE{}: expecting \';\'.format(line))
        else:
            print('写语句 LINE{}: expecting \')\'.format(line))
    else:
        print('写语句 LINE{}: undefined variable
\{ }\'.format(line,get_symbol(idx)))
    else:
        print('写语句 LINE{}: invalid variable name
\{ }\'.format(line,get_symbol(idx)))
    else:
        print('写语句 LINE{}: expecting \'(\'.format(line))
    else:
        return

```

15. 算术表达式()

算术表达式是形如 $xx-xx*xx$ 的短语，由项组成，算法实现为利用子函数项()内部的右递归进行表达式的拓展。

```

def 算术表达式():
    global idx
    global line

```



```

项()
if get_tag(idx)==18:
    advance()
    算术表达式()
# elif get_tag(idx)==23:
#     advance()
return

```

16. 项()

项表示 xx 、 $xx*xx$ 等短语，在分析时，通过区分项与因子，达到区分 $*$ 与 $-$ 两种不同级别运算符优先级的目的。项()中存在递归，用于向右递归拓展，可以实现连续加减。

```

def 项():
    global idx
    因子()
    if get_tag(idx)==19:
        advance()
        项()
    else:
        return

```

17. 因子()

因子是不会被再分的算术成员，可能是数字、标识符或者函数调用。

```

def 因子():
    global idx
    # print(var_list)
    # print(get_symbol(idx))
    if get_tag(idx)==11:
        advance()
    elif get_symbol(idx) in var_list[0] or get_symbol(idx) in pvar_list:
        advance()
    elif get_symbol(idx) in pro_list:
        函数调用()
    else:
        print('因子 LINE{}: Invalid coefficient in the formula.'.format(line))

```

18. 条件语句()

条件语句用于对条件判断语句进行分析，通过嵌套 if-else 检查语法，在其中调用条件表达式，并通过执行语句()进行递归下降分析。

```

def 条件语句():
    global idx
    global line
    if get_tag(idx)==4:

```

```

    advance()
    条件表达式()
    if get_tag(idx)==5:
        advance()
        执行语句(1)
        if get_tag(idx)==6:
            advance()
            执行语句(1)
        else:
            return
    else:
        print('条件语句 LINE{}: missing execution instructions.'.format(line))
else:
    return

```

19. 条件表达式()

条件表达式()用于递归下降分析条件语句中的条件表达式，形如 $xx < \text{比较符} > xx$ ， xx 为算术表达式。

```

def 条件表达式():
    global idx
    算术表达式()
    if 12<=get_tag(idx)<=17:
        advance()
        算术表达式()
    else:
        return

```

20. 主函数

主函数主要用于文件处理，打开文件并将上游文件内容读取存入全局列表中用于语法分析。然后调用开始()开始分析。

```

if __name__=='__main__':

    input=open('sample.dyd','rb')
    var_file=open('sample.var','w')
    pro_file=open('sample.pro','w')
    error_file=open('sample.dys','w')

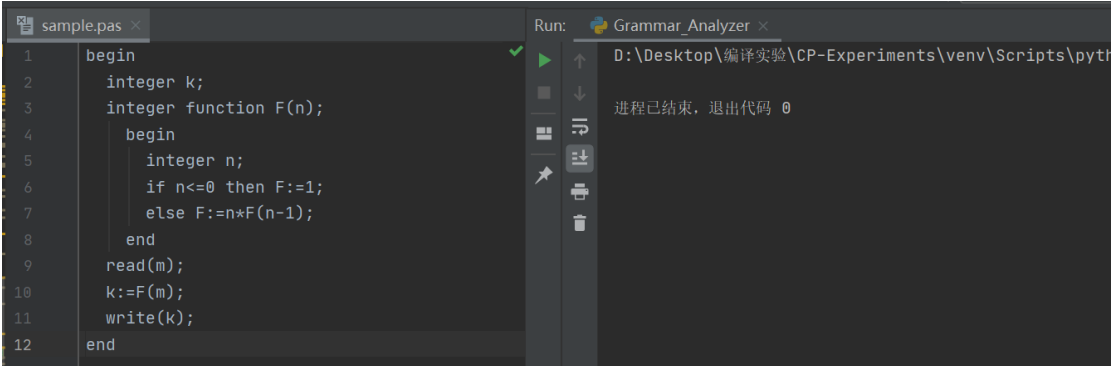
    vector=input.readline()
    while vector:
        vector_list.append(vector)
        vector=input.readline()

    开始()

```

九、实验数据及结果分析

1. 示例 1（正确代码）：



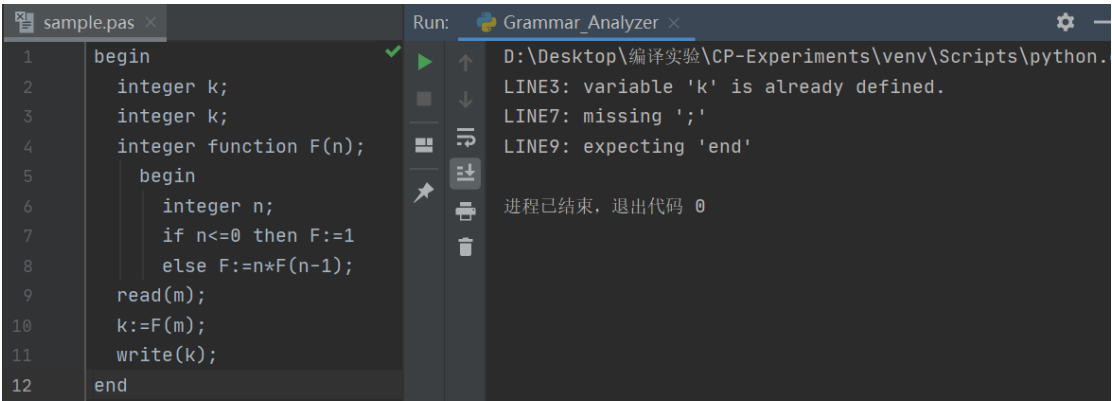
```
sample.pas x
1 begin
2   integer k;
3   integer function F(n);
4     begin
5       integer n;
6       if n<=0 then F:=1;
7       else F:=n*F(n-1);
8     end
9   read(m);
10  k:=F(m);
11  write(k);
12 end
```

Run: Grammar_Analyzer x
D:\Desktop\编译实验\CP-Experiments\venv\Scripts\python.
进程已结束，退出代码 0

如图，给定的代码为不含语法错误的代码，语法分析完成，无报错信息。函数列表和变量列表结果如图，符合预期结果。

```
变量列表: k,m
函数列表: F(n)
```

2. 示例 2（错误代码）：



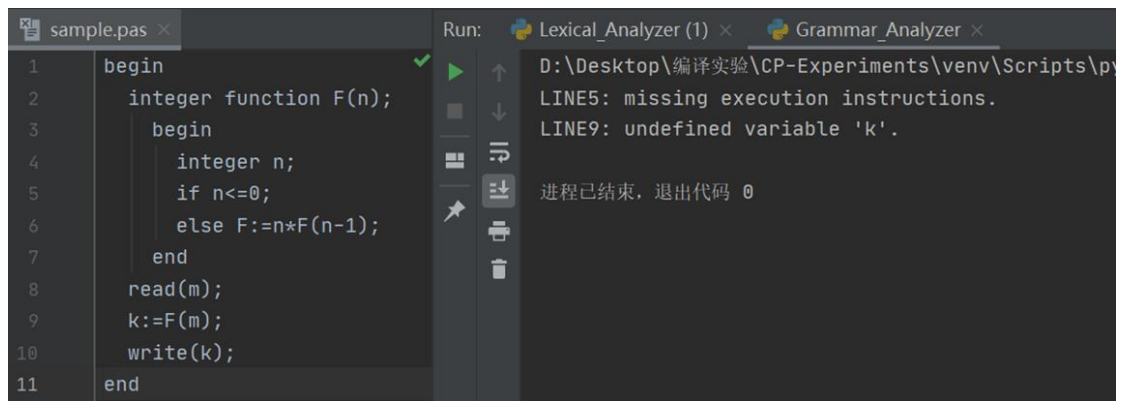
```
sample.pas x
1 begin
2   integer k;
3   integer k;
4   integer function F(n);
5     begin
6       integer n;
7       if n<=0 then F:=1
8       else F:=n*F(n-1);
9     read(m);
10    k:=F(m);
11    write(k);
12 end
```

Run: Grammar_Analyzer x
D:\Desktop\编译实验\CP-Experiments\venv\Scripts\python.
LINE3: variable 'k' is already defined.
LINE7: missing ';' ;
LINE9: expecting 'end'
进程已结束，退出代码 0

如图，该示例代码中重复定义变量 k，在函数定义中，赋值语句后缺少“;”，函数定义末尾缺少“end”。语法分析器对源程序进行语法分析，一次性给出了所有语法错误的行数及出错类型。但该错误并不影响函数和变量的定义，因此变量列表和函数列表依然能够正确生成，如图。

```
变量列表: k,m
函数列表: F(n)
```

3. 示例 3（错误代码）：



```
sample.pas x
1 begin
2   integer function F(n);
3   begin
4     integer n;
5     if n<=0;
6     else F:=n*F(n-1);
7   end
8   read(m);
9   k:=F(m);
10  write(k);
11 end
```

Run: Lexical_Analyzer (1) x Grammar_Analyzer x

D:\Desktop\编译实验\CP-Experiments\venv\Scripts\python.exe D:\Desktop\编译实验\CP-Experiments\venv\Scripts\python.exe D:\Desktop\编译实验\CP-Experiments\venv\Scripts\python.exe

LINE5: missing execution instructions.

LINE9: undefined variable 'k'.

进程已结束，退出代码 0

如图，该示例代码中未定义变量 `k` 但在第 9 行对 `k` 进行赋值，第 5 行的条件判断后缺乏执行语句。语法分析器对源程序进行语法分析，一次性给出了所有语法错误的行数及出错类型。变量列表和函数列表如图。

```
变量列表: m
函数列表: F(n)
```

十、实验结论

本实验程序较好地完成了递归下降分析器的设计与实现，能够对所给文法的程序进行语法分析，生成变量名表和过程名表，如果源程序有语法错误则给出出错类型及所在行数。

十一、总结及心得体会

利用 Python 实现了实现求 $n!$ 的极小语言的语法分析程序，能正确有效地运行，分析 `dyd` 文件，判断源代码中是否有语法错误，并指出这些错误。通过本次实验，我对语法分析器有了更加深刻的认识，对编译器编译源代码生成目标代码的第二个关键步骤有了更深入的掌握和更透彻的理解。同时，通过自己编写代码实现该语法分析器，提高了自己的逻辑分析能力和编码能力。

十二、对本实验过程及方法、手段的改进建议

该程序整体上满足实验要求，能正确识别出多种语法错误，但仍存在一些问题，如：

1. 测试过程中发现存在某些无效输入会导致语法分析程序本身报错，此问题暂时并未解决；
2. 报错信息过于机械，有时不能准确表达代码中的错误，比如在 `if-then` 的 `then` 前加“;”，理想的报错结果应该是直接判定为重大错误直接退出，

但该程序会报“缺少执行语句”和“缺少 end”。除此之外还有其它类似问题，该程序还有待优化。

报告评分：

指导教师签字：

本实验参考源代码如下：

实验一：

```
symbol_list=[0, 'begin', 'end', 'integer', 'if', 'then', 'else', 'function', 'read', 'write',
10, 11, '=', '<>', '<=', '<', '>=', '>', '-', '*', ':=', '(', ')', ';']

if __name__=='__main__':

    input=open('sample.pas','rb')
    output=open('sample.dyd','w')
    error=open('sample.err','w')

    idx=0
    line=1

    while True:

        error_type=0
        ch=input.read(1)
        idx+=1
        temp=b''

        if ch==b' ':           # 空格
            continue

        elif ch==b'\n' or ch==b'\r':       # 换行
            ch=input.read(1)
            temp=b'\n'
            line+=1
            idx=0
            output.write('{} {} \n'.format('EOLN'.rjust(16, ' '), 24))
            continue
```

```

elif ch==b'':          # 文件结束
    temp=b''
    output.write('{} {} \n'.format('EOF'.rjust(16, ' '), 25))
    break

elif ch.isalpha():     # 标识符
    while ch.isalnum():
        temp+=ch
        idx+=1
        ch=input.read(1)
        if len(temp)>16:
            error_type=3
            continue
    if ch != b'':
        input.seek(-1, 1)
        idx-=1

elif ch.isdigit():     # 常数
    while ch.isdigit():
        temp+=ch
        idx+=1
        ch=input.read(1)
        input.seek(-1,1)
        idx-=1

elif ch in [b'-' , b'*' , b'(' , b')' , b';']:
    temp=ch

elif ch==b':':
    ch=input.read(1)
    idx+=1
    if ch==b'=':
        temp=b':='
    else:
        if ch!=b'':
            input.seek(-1,1)
            idx-=1
        error_type=1

elif ch==b'>':
    ch=input.read(1)
    idx+=1
    if ch==b'=':

```

```

        temp=b'>='
    else:
        if ch!=b'':
            input.seek(-1,1)
            idx-=1

    elif ch==b'<':
        ch=input.read(1)
        idx+=1
        if ch==b'=':
            temp=b'>='
        else:
            if ch!=b'':
                input.seek(-1,1)
                idx-=1

    else:
        error_type=2
        # continue

# 分析
temp=temp.decode()
if temp.isdigit():
    output.write('{} {} \n'.format(temp.rjust(16, ' '),11))
    continue

if temp in symbol_list:
    symbol_index=symbol_list.index(temp)
    output.write('{} {} \n'.format(temp.rjust(16, ' '),symbol_index))
else:
    output.write('{} {} \n'.format(temp.rjust(16, ' '),10))

if error_type:
    if error_type==1:
        error.write('LINE:{} 冒号不匹配 \n'.format(line))
    elif error_type==2:
        ch=ch.decode()
        error.write('LINE:{} 非法字符\''{}'\n'.format(line,ch))
    elif error_type==3:
        error.write('LINE:{} 标识符\''{}'\n长度溢出 \n'.format(line,temp))

input.close()

```

```
output.close()
error.flush()
```

实验二：

```
global idx
global line
global vector_list
global pro_list
global pvar_list
global var_list

idx=0
line=1
vector_list=[]
pro_list=[]
pvar_list=[]
var_list=[[[]],[]]]

def advance():
    global idx
    global line
    idx+=1
    if get_tag(idx)==24:
        line+=1
        idx+=1

def get_tag(idx):
    global vector_list
    vector=vector_list[idx]
    symbol_tag=vector.split()
    return int(symbol_tag[1].decode())

def get_symbol(idx):
    global vector_list
    vector=vector_list[idx]
    symbol_tag=vector.split()
    return symbol_tag[0].decode()

def 定义了变量(var,flag): #var 为变量名, flag 用来区分主函数的变量和函数内的临时变量
    global var_list
    global pro_list
    global line
    if flag==0:
```



```

        if var in var_list[0]:
            print('LINE{}: variable \'{}\'' is already defined.'.format(line,var))
        else:
            var_list[0].append(var)
            var_file.write('[int]{}\n'.format(var))
elif flag==1:
    if var in var_list[1]:
        print('LINE{}: variable \'{}\'' is already defined.'.format(line,var))
    else:
        var_list[1].append(var)
        pro_file.write('[int]{}\n'.format(var))

def 定义了函数(pro_name,pro_var_list):
    global pro_list
    global pvar_list
    pro_list.append(pro_name)
    # pvar_list.append(pro_name)
    pvar_list.append(pro_var_list)
    pro_file.write('[int]{}({})\n'.format(pro_name,pro_var_list))
    # for i in range(len(pro_var_list)):
    #     pro_file.write(pro_var_list[i])
    #     if i!=len(pro_var_list)-1:
    #         pro_file.write(',')
    # pro_file.write(')\n')

def 开始():
    global idx
    global line
    if get_tag(idx)==1:
        advance()
        说明语句表()
        执行语句表(0)
        # print(idx)
    if get_tag(idx)==2:
        return
    else:
        print('LINE{}: expecting \'end\'''.format(line))
        advance()
    else:
        print('LINE{}: missing \'begin\'''.format(line))
        advance()

def 说明语句表():
    变量说明(0)

```

函数说明()

```
def 变量说明(flag):
    global idx
    global line
    if get_tag(idx)==3:
        advance()
        symbol=get_symbol(idx)
        if get_tag(idx)==10:
            定义了变量(symbol,flag)
            advance()
            if get_tag(idx)==23:
                advance()
                变量说明(flag)
            else:
                print('LINE{}: missing \';\''.format(line-1))
        elif get_tag(idx)==7:
            idx-=1
            return
        else:
            print('LINE{}: invalid variable name \'{ }\'''.format(line,symbol))
            advance()
            if get_tag(idx)==23:
                变量说明(flag)
            else:
                print('LINE{}: missing \';\''.format(line-1))
    else:
        return
```

```
def 函数说明():
    函数声明()
    函数过程()
```

```
def 函数声明():
    global idx
    global line
    if get_tag(idx)==3:
        advance()
    if get_tag(idx)==7:
        advance()
    if get_tag(idx)==10:
        pro_name=get_symbol(idx)
        advance()
    if get_tag(idx)==21:
```

```

        advance()
        if get_tag(idx)==10:
            var_name=get_symbol(idx)
            定义了函数(pro_name,var_name)
            advance()
            if get_tag(idx)==22:
                advance()
                if get_tag(idx)==23:
                    advance()
                else:
                    print('LINE{}: missing \';\''.format(line))
            else:
                print('LINE{}: expecting \')\''.format(line))
        elif get_tag(idx)!=22:
            print('LINE{}: invalid parameter
\'{}\''.format(line,get_symbol(idx)))
            advance()
        else:
            var_name='null'
            advance()
        else:
            print('LINE{}: Syntax error.')
            advance()
        else:
            print('LINE{}: invalid function name \'{}\''.format(line,get_symbol(idx)))
            advance()
        else:
            print('LINE{}: Syntax error.'.format(line))
            advance()
    else:
        return

def 函数过程():
    global idx
    global line
    if get_tag(idx)==1:
        advance()
        变量说明(1)
        执行语句表(1)
    if get_tag(idx)!=2:
        print('LINE{}: expecting \'end\''.format(line))
    else:
        advance()
    return

```

```

def 执行语句表(flag):
    执行语句(flag)

def 执行语句(flag):
    读语句()
    赋值语句(flag)
    写语句()
    条件语句()

def 读语句():
    global idx
    if get_tag(idx)==8:
        advance()
    if get_tag(idx)==21:
        advance()
    if get_tag(idx)==10:
        var_name=get_symbol(idx)
        定义了变量(var_name, 0)
        定义了变量(var_name, 1)
        advance()
    if get_tag(idx)==22:
        advance()
    if get_tag(idx)==23:
        advance()
        读语句()
    else:
        print('LINE{}: missing \';\''.format(line-1))
    else:
        print('LINE{}: expecting \')\''.format(line))
    else:
        print('LINE{}: invalid variable name \'{\'\''.format(line,get_symbol(idx)))
    else:
        print('LINE{}: expecting \'(\'\''.format(line))
    else:
        return

def 赋值语句(flag):
    global idx
    global line
    global pvar_list
    if get_tag(idx)==10:
        if flag==0:
            if get_symbol(idx) in var_list[0]:

```

```

        advance()
        if get_tag(idx)==20:
            advance()
            算术表达式()
            if get_tag(idx)==23:
                advance()
                赋值语句(flag)
            else:
                print('LINE{}: missing \';\'' .format(line-1))
        else:
            print('LINE{}: Syntax error.' .format(line))
    else:
        print('LINE{}: undefined variable \'{ }\'' .format(line,get_symbol(idx)))
else:
    if get_symbol(idx) in pro_list or get_symbol(idx) in pvar_list:
        advance()
        if get_tag(idx)==20:
            advance()
            算术表达式()
            if get_tag(idx)==23:
                advance()
                赋值语句(flag)
            else:
                print('LINE{}: missing \';\'' .format(line-1))
        else:
            print('LINE{}: Syntax error.' .format(line))
    else:
        print('LINE{}: undefined variable \'{ }\'' .format(line,get_symbol(idx)))
else:
    return

def 写语句():
    global idx
    global line
    if get_tag(idx)==9:
        advance()
        if get_tag(idx)==21:
            advance()
            if get_tag(idx)==10:
                if get_symbol(idx) in var_list[0]:
                    advance()
                if get_tag(idx)==22:
                    advance()
                if get_tag(idx)==23:

```

```

        advance()
        写语句()
    else:
        print('LINE{}: expecting \';\''.format(line))
    else:
        print('LINE{}: expecting \')\''.format(line))
    else:
        print('LINE{}: undefined variable \'{\''.format(line,get_symbol(idx)))
    else:
        print('LINE{}: invalid variable name \'{\''.format(line,get_symbol(idx)))
    else:
        print('LINE{}: expecting \'(\''.format(line))
else:
    return

def 算术表达式():
    global idx
    global line
    项()
    if get_tag(idx)==18:
        advance()
        算术表达式()
    # elif get_tag(idx)==23:
    #     advance()
    return

def 项():
    global idx
    因子()
    if get_tag(idx)==19:
        advance()
        项()
    else:
        return

def 因子():
    global idx
    # print(var_list)
    # print(get_symbol(idx))
    if get_tag(idx)==11:
        advance()
    elif get_symbol(idx) in var_list[0] or get_symbol(idx) in pvar_list:
        advance()
    elif get_symbol(idx) in pro_list:

```

```

        函数调用()
    else:
        print('LINE{}: Invalid coefficient in the formula.'.format(line))

def 函数调用():
    global idx
    global line
    advance()
    if get_tag(idx) == 21:
        advance()
        if get_tag(idx) == 10:
            算术表达式()
            if get_tag(idx) == 22:
                advance()
                # if get_tag(idx) == 23:
                #     advance()
                #     赋值语句(1)
                # else:
                #     print('LINE{}: expecting ';'.'.format(line))
            else:
                print('LINE{}: expecting ')'.'.format(line))
        else:
            print('LINE{}: invalid variable name {}'.format(line, get_symbol(idx)))
    else:
        print('LINE{}: expecting '('.'.format(line))

def 条件语句():
    global idx
    global line
    if get_tag(idx)==4:
        advance()
        条件表达式()
        if get_tag(idx)==5:
            advance()
            执行语句(1)
            if get_tag(idx)==6:
                advance()
                执行语句(1)
            else:
                return
        else:
            print('LINE{}: missing execution instructions.'.format(line))
    else:
        return

```

```
def 条件表达式():
    global idx
    算术表达式()
    if 12<=get_tag(idx)<=17:
        advance()
        算术表达式()
    else:
        return

if __name__=='__main__':

    input=open('sample.dyd','rb')
    var_file=open('sample.var','w')
    pro_file=open('sample.pro','w')
    error_file=open('sample.dys','w')

    vector=input.readline()
    while vector:
        vector_list.append(vector)
        vector=input.readline()

    开始()
```