编译原理实验一实验报告

递归下降的分析处理程序

课程名称：编译原理
题目名称：递归下降的分析处理程序
学生学院：计算机科学与技术学院
小组成员：朱河勤, 张世聪, 徐瑞, 詹慧悠
指导教师：郑启龙

一、实验目的

本次实验的目的是针对给定的简化版 C 语言声明文法，实现一个递归下降的分析处理程序。

递归下降分析法是语法分析中最易懂的一种方法，基本原理是：对每个非终结符号（分别代表一个语法单位）按其产生式结构构造相应语法分析子程序，以完成该非终结符号所对应的语法单位的分析和识别任务。其中终结符号产生匹配命令，而非终结符号则产生过程调用命令。因为文法可以递归，相应子程序也是递归的，所以称这种方法为递归子程序下降法或递归下降法。其中子程序的结构与产生式结构几乎是一致的。

二、实验原理

具体而言，本次实验我们主要分成了两块来实现：词法分析、语法分析。以下分别进行介绍。具体的实验代码用 Python 语言来进行编写。

本次实验的实现共由使用了这样的架构：

```
|--- declarationParser.py
    |--- token_scanner.py
```

词法分析在文件 token_scanner.py 中实现，语法分析和输入输出界面在 declarationParser.py 中实现。其中 declarationParser.py 需要调用 token_scanner.py 作为头文件来处理词法。

以下具体介绍各个模块的实现。

1. 词法分析

我们的词法分析具体如下表所示。

Lexical unit	**Regular expression**
NAME	[a-zA-Z_][a-zA-Z_0-9]*
NUM	\d*\.\d+|\d+
POINTER	*
COMMA	\,
SEMICOLO	\;
VOID	void
INT	int
LEFT	\(
RIGHT	\)
L2	\[
R2	\]
WS	\s+

词法分析的实现是在 token_scanner.py 的 gen_token 函数中。该函数可以对一段文本以 WS 类字符为分隔进行词法分析。

2. 语法分析

该部分的实验思路如下：

1. 根据 C 语言的语法规则，写出 C 语言变量声明的上下文无关文法；
2. 求每个产生式的 Select 集：
3. 判断是否满足递归下降法分析条件，若不满足用消除左递归和提取公因子等文法等价变换操作对文法进行变换，使其满足递归下降法的要求；
4. 构造递归下降语法分析程序，对文法中的每个非终结符号按其产生式结构产生相应的语法分析子程序,完成相应的识别任务。其中终结符号产生匹配命令，非终结符号则产生调用命令。实际的语法分析工作从调用主程序开始，根据产生式递归调用各个分析子程序。

1.
我们的语法分析根据以下的 c 语法规则来进行。

```scala
translation_unit
  : declaration
  | translation_unit declaration
      ;

declaration
  : declaration_specifiers init_declarator_list ';'
      ;

declaration_specifiers
  : type_specifier
  ;

init_declarator_list
  : init_declarator
  | init_declarator_list ',' init_declarator
  ;

init_declarator
  : declarator
  ;

type_specifier
  : VOID
  | INT
  ;

declarator
  : pointer direct_declarator
  | direct_declarator
```

```
  ;

direct_declarator
  : IDENTIFIER
  | '(' declarator ')'
  | direct_declarator '[' CONSTANT_INT ']'
  | direct_declarator '(' parameter_type_list ')'
  | direct_declarator '(' ')'
  ;
 pointer
  : '*'
  | '*' pointer
  ;

parameter_type_list
  : parameter_list
      ;

parameter_list
  : parameter_declaration
  | parameter_list ',' parameter_declaration
  ;

parameter_declaration
  : declaration_specifiers declarator
  ;
```

2.根据上面的语法式，我们可以看出 Select 集。

3.判断是否满足递归下降法分析条件，若不满足用消除左递归和提取公因子等文法等价变换操作对文法进行变换，使其满足递归下降法的要求；
从中看出，Translation_unit、init_declaration_list、direct_declarator、parameter_list 需要消除左递归。接下来我们就可以根据规则来进行编写代码进行分析了。

4.根据前面的步骤，我们可以得到代码

```
classdeclarationParser(parser):
    type_size = {'INT':1,'POINTER':1,'VOID':1}
defstatement(self):
'''non-terminate-symbol: translation_unit'''
whileself.i<self.n:
self.declaration()
defdeclaration(self):
        symType = self.declaration_specifiers()
self.init_declarator_list(symType)
self.match('SEMICOLON')
defdeclaration_specifiers(self):
returnself.type_specifier()
deftype_specifier(self):
returnself.match().value
definit_declarator_list(self,symType):
while1:
self.init_declarator(symType)
```

```
ifself.isType('COMMA'):
self.match()
else:break
definit_declarator(self,symType):
        s = self.declarator(symType)
print(s)
defdeclarator(self,symType):
        np = self.pointer() # np>=0
        s = self.direct_declarator(symType,np)
return s
defdirect_declarator(self,symType,np):
        tp = 'pointer(' *np + symType + ')'*np
        args =''
        inner = '$'
        name = ''
ifself.isType('NAME'):
            name = self.match().value+'::'
elifself.isType('LEFT'): # (
self.match()
            inner = self.declarator('$') #mark
self.match('RIGHT')
ifself.isType('LEFT'):
self.match()
            li = ['void']
ifnotself.isType('RIGHT'):
                li = self.parameter_type_list()
self.match('RIGHT')
ifself.isType('L2'):
raiseException('Array of Functions is not allowed')
            args = ' X '.join(li)
elifself.isType('L2'):
            li = []
whileself.isType('L2'):
self.match()
assertself.isType('NUM')
                li.append(int(self.match().value))
self.match('R2')
ifself.isType('LEFT'):
raiseException('Array of Function can not be returned from functions')
for i inreversed(li):
                tp = 'array({},{})'.format(i,tp)
if args!='':
            tp = 'function( {args} =>{tp})'.format(args=args,tp =tp )
return name+inner.replace('$',tp)

defpointer(self):
        n = 0
whileself.isType('POINTER'):
            n+=1
self.match('POINTER')
return n
defparameter_type_list(self):
returnself.parameter_list()
defparameter_list(self):
        li = []
while1:
```

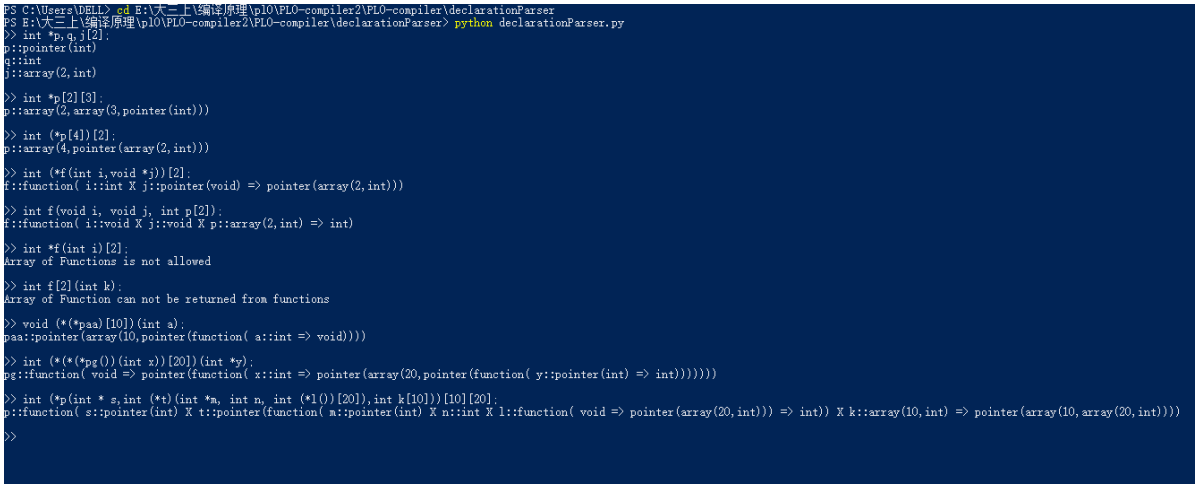
```
            argType = self.parameter_declaration()
            li.append(argType)
ifself.isType('COMMA'):
self.match()
else:break
return li
defparameter_declaration(self):
        symType = self.declaration_specifiers()
returnself.declarator(symType)
```

三、结果分析

（测试样例）

```
## test.txt
int *p,q,j[2];
int *p[2][3];
int (*p[4])[2];
int (*f(int i,void *j))[2];
int f(void i, void j, int p[2]);
//wrong
int *f(int i)[2];
int f[2](int k);
void (*(*paa)[10])(int a);
int (*(*(*pg())(int x))[20])(int *y);
int (*p(int * s,int (*t)(int *m, int n, int (*l())[20]),int k[10]))[10][20];
```

（运行结果）



详情为：

```
PS E:\大三上\编译原理\pl0\PL0-compiler2\PL0-compiler\declarationParser> python
declarationParser.py
>> int *p,q,j[2];
p::pointer(int)
q::int
j::array(2,int)

>> int *p[2][3];
p::array(2,array(3,pointer(int)))

>> int (*p[4])[2];
```

```
p::array(4,pointer(array(2,int)))

>> int (*f(int i,void *j))[2];
f::function( i::int X j::pointer(void) => pointer(array(2,int)))

>> int f(void i, void j, int p[2]);
f::function( i::void X j::void X p::array(2,int) => int)

>> int *f(int i)[2];
Array of Functions is not allowed

>> int f[2](int k);
Array of Function can not be returned from functions

>> void (*(*paa)[10])(int a);
paa::pointer(array(10,pointer(function( a::int => void))))

>> int (*(*(*pg())(int x))[20])(int *y);
pg::function( void => pointer(function( x::int =>
pointer(array(20,pointer(function( y::pointer(int) => int)))))))

>> int (*p(int * s,int (*t)(int *m, int n, int (*l())[20]),int k[10]))[10][20];
p::function( s::pointer(int) X t::pointer(function( m::pointer(int) X n::int X
l::function( void => pointer(array(20,int))) => int)) X k::array(10,int) =>
pointer(array(10,array(20,int))))

>>
```

四、总结

本次实验成功根据给定的简化版 C 语言声明文法，实现一个递归下降的分析处理程序。通过此次实验，我们进一步理解了语法分析在编译程序中的作用，以及它与词法分析程序的关系，还掌握了递归下降语法分析方法的主要原理和文法变换方法。此外还进一步理解了递归下降分析法对文法的要求。