编译原理 Lab2 实验报告

余旻晨　　141250177

1. 实验介绍

本次实验旨在完成编译中的语法分析部分，写一个自己的 Yacc，并对程序进行语法分析。在这次实验中，Yacc 采用了 LL(1)文法进行语法分析。Yacc 的作用是根据文法定义生成对应的语法分析表。最后利用分析表完成语法分析程序的编写。

2. 方法说明

2.1 Yacc 实现部分

Yacc 的实现在代码中放置于 yacc 包下的 YaccAnalyzer 类中，用到的数据结构放置在 util 包下。Yacc 的输入文件为 doc/definition.txt。

2.1.1 输入定义

definition.txt 中定义了终止符和非终止符集合，分别声明为 Vt 和 Vn。接下来声明了 LL(1)文法，以%%为分隔符。如截图所示。

```
Vt:id,num,if,(,),{,},else,while,+,*,=,==,||,&&,e
Vn:S,E,R,T,Y,F,C,V,Z,X,B,L,
%%
S:id=E
S:if(C){S}else{S}
S:while(C){S}
E:TR
R:+TR
R:e
T:FY
Y:*FY
Y:e
F:(E)
F:L
C:ZV
V:||ZV
V:e
Z:BX
X:&&BX
X:e
B:(C)
B:L==L
L:id
L:num
```

该文法由三部分组成。起始符 S 可以产生赋值语句，if-else 条件选择语句和 while 循环语句三大类语句。L 表示标识符(id)或者数字(num)。

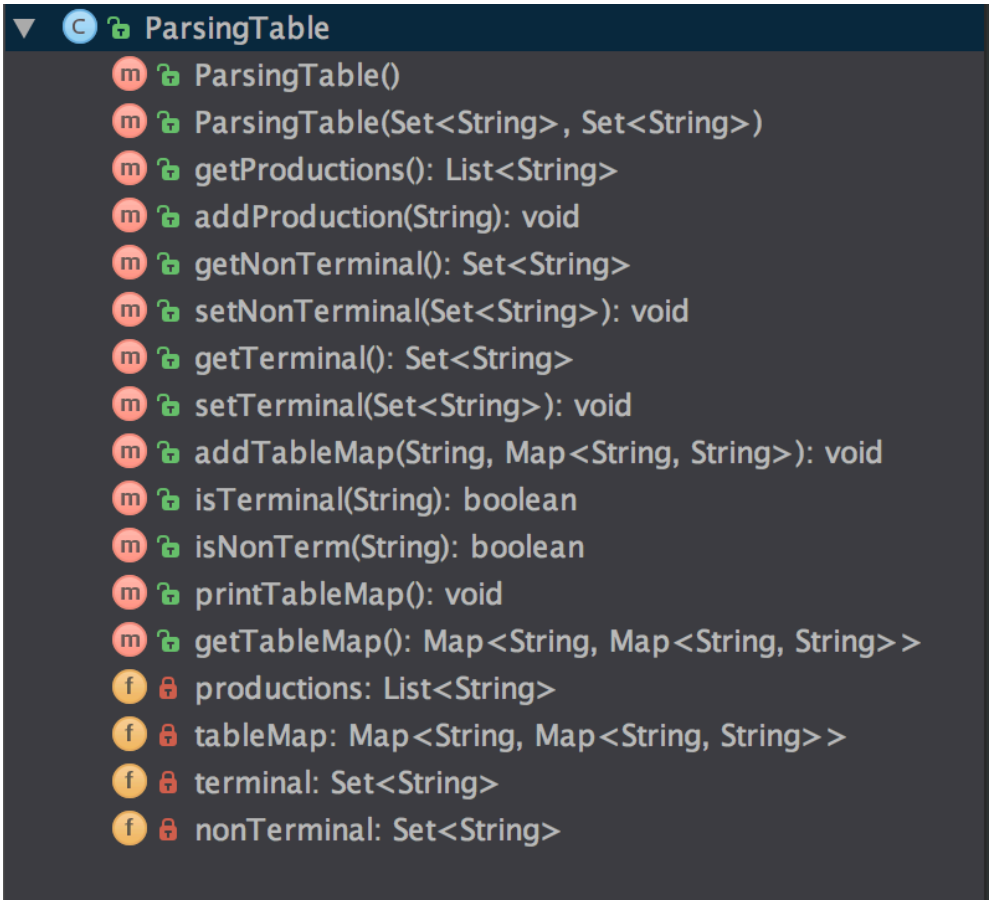
对于赋值语句，相关符号有+，*，()以及 id 和 num。规定括号的优先级大于乘号的优先级大于加号的优先级，且乘号和加号符合左结合的运算规则。该部分的非终止符包含 S，E，R，T，Y，L。符合 LL(1)文法。

对于条件选择语句，为了避免二义性，文法中的 if 和 else 成对出现，且判断符仅采用==符号。文法对复杂的逻辑判断提供了与(&&)和或(||)，并规定了优先级与大于或。同样，括号的优先级最大。条件判断部分的非终止符包含 S，C，Z，V，X，B，L。符合 LL(1)文法。

对于循环语句，关键字 while，条件判断部分同上。

2.1.2 数据结构说明

在 util 包放置所用的数据结构。除了词法分析程序所用的数据结构之外，增加了 ParsingTable 这一数据结构，用于记录由 Yacc 生成的语法分析表 。ParsingTable 的数据结构如下。



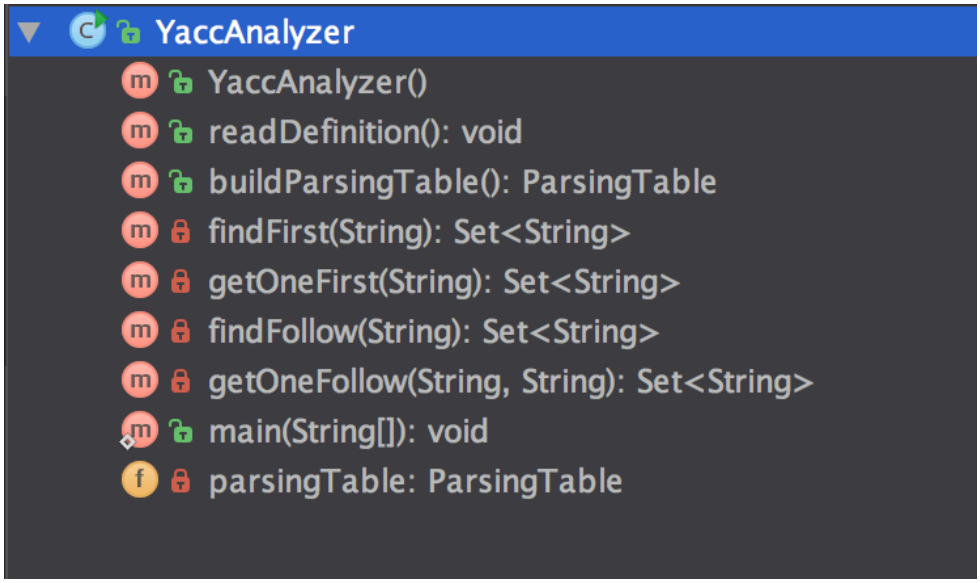
ParsingTable 的属性有终止符集合 terminal，非终止符集合 nonTerminal，产生式列表 productions 以及实际的语法分析表 tableMap。tableMap 是以 String 为 key，Map 为 value 的映射，key 存放了分析表中的非终止符，即每一个键值对对应分析表中的一行。value 是一个非终止符对应产生式的映射，即 value 中的 key 表示一列，一个键值对就对应一

个表格项。同时定义了与属性相关的行为，除了 getter 和 setter 外，还有判断是否为终止符或非终止符以及打印语法分析表的方法。具体实现参见代码。

此外，为了方便语法分析，在 Type 枚举类中添加了 END 枚举，用于表示$。

2.1.3 实现说明

Yacc 的实现参见 YaccAnalyzer 类。如图所示，主要方法有读取定义文件，创建 ParsingTable 以及计算 First 和 follow 值。



生成语法分析表的第一步是从 definition.txt 中读取相关定义，用于初始化 ParsingTable，接着开始生成 ParsingTable 的核心部分 tableMap。需要对每一个产生式计算在语法分析表中的对应的位置。在方法中遍历产生式，若是 epsilon 产生式则计算 Follow，其他产生式计算 First。

计算 First 和 Follow 采用递归的方法实现。

对于 First，先分析产生式右部的首可识别字符，若是终止符则完成计算直接返回。若是非终止符则计算该非终止符包含产生式的 First 值。再判断集合中是否含有 epsilon，若不含，则可直接返回计算得到的 First 值，若含有，则需要出去 epsilon，同时进行下一个可识别字符的 First 计算。以此类推，直到找到不含 epsilon 的非终止符产生式。最后返回所有得到的 First 值。

对应 Follow，需要对每个在右部有原非终止符的产生式进行 Follow 计算。考虑到集合运算的结果集最小化，故对于 A=AUB 的情形视为 A=B。计算思路如下：若原非终止符的右边可识别字符为终止符，则直接返回该终止符。若原非终止符的右边可识别字符为非终止符，则计算该非终止符的 First 值，再判断是否含有 epsilon；若不含有，则该 First 值即为这个产生式对应原非终止符的 Follow 值；若含有，则将去除 epsilon 的 First 集合加入 Follow 集合，再计算该非终止符的 Follow 值，加入返回的 Follow 值集合。若原非终止符右部无可识别字符，则计算产生式左部的 Follow

值作为返回结果。此时要考虑左部的非终止符和要求的非终止符相等而出现无穷递归陷入死循环的情形。所以出现这种情况，按照结果集最小化的原则，直接返回空集合。

2.2 语法分析程序部分

语法分析部分的代码放置在 main 包的 Main 类中，分析中用到了 Yacc 阶段生成的语法分析表。

2.2.1 语法分析说明

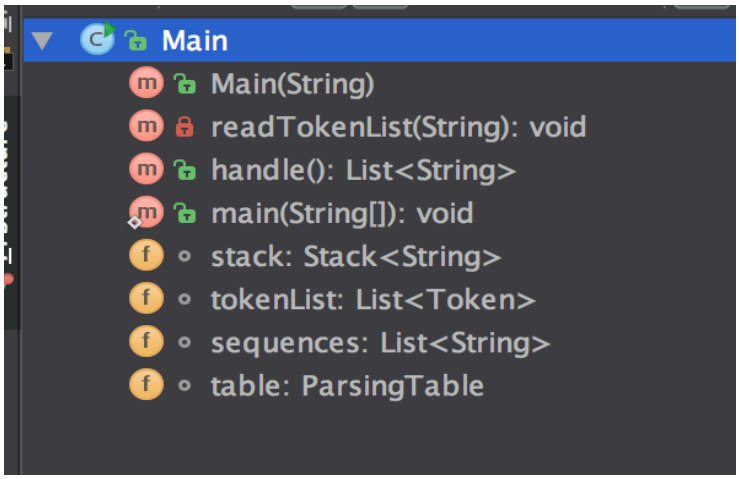
在本次实验中，语法分析的目的是为了基于在 definition.txt 中声明的文法，生成一段代码的语法分析序列。为了同第一次实验中的词法分析程序相结合，这里采用了数据耦合的松耦合方式。具体思路为：对一段程序，先用词法分析程序分析得到所有的 Token，将这些 Token 按顺序存入对应的文件中。语法分析程序只需要从文件中将所有 Token 按序读取，即可对这段代码进行分析。这样就可以利用最小词法单元，避免了对源代码的重复解析。

接下来，就是根据 Token 序列和 Yacc 生成的语法分析表进行语法分析。首先需要构造一个栈，将栈底放入$和开始符 S。然后根据栈顶元素和 Token 序列找语法分析表中对应的表格项。对产生式右部进行分割与压栈，然后继续进行匹配。当栈顶的非终止符与当前 Token 相同时，得到下一个 Token 并将栈顶元素移出。知道最后的$Token 和栈中的$重合，分析结束。

当语法分析成功完成时，在控制台出入 success，并将分析序列打出。当代码不符合语法时，语法分析程序输出错误信息，并将错误的行号打出，同时输出分析到错误行前的分析序列。由于在词法分析中，Token 的第三个元素即为所在的行数，故可以很容易地得到当前程序错误的行数。

2.2.2 程序实现

语法分析程序的结构如下。



如图所示，语法分析程序的核心方法是 handle。在 handle 中，对 Token 序列和栈元素进行基于语法分析表的操作。得到 Token 的 List 序列的迭代器 Iterator，通过 next()操作实现读头的变化。在程序中需要对当前栈顶元素是否为终止符进行判断。若是终止符，则与当前 Token 进行匹配，成功则移动读头，移除栈顶元素，否则终止语法程序并报错。若是非终止符，则在语法分析表中找对应当前 Token 的产生式，若找不到则报错。找到对应的产生式后将元素按从右到左的顺序压入栈内。此时有两类较为特殊的情况需要考虑。第一种是赋值符号=和比较相等符号==需要避免程序判断错误。第二类是如果对应的产生式为 epsilon 产生式时，不进行压栈操作。在进行匹配的时候需要注意在 Token 为标识符(id)和数字(num)这两种情形下，需要对其 Type 进行匹配，其余情况对 Token 的 value 进行匹配。

3. 测试展示

3.1 Yacc 展示

Yacc 的输入为 definition.txt 文件，内容见 2.1.1 输入定义的截图。得到最终结果为语法分析表，输出在控制台的结果如下：

```
0   S:id=E
1   S:if(C){S}else{S}
2   S:while(C){S}
3   E:TR
4   R:+TR
5   R:e
6   T:FY
7   Y:*FY
8   Y:e
9   F:(E)
10  F:L
11  C:ZV
12  V:||ZV
13  V:e
14  Z:BX
15  X:&&BX
16  X:e
17  B:(C)
18  B:L==L
19  L:id
20  L:num
    ==  ||  &&  num (   )   *   +   wh  el  id  {   if  }   =   $
R                   5       4                       5       5
B               18  17                  18
S                                   2   0       1
C               11  11                  11
T               6   6                   6
E               3   3                   3
F               10  9                   10
V       12              13
X       16  15          16
Y                       8   7   8                   8       8
Z               14  14                  14
L               20                      19
```

为了显示方便，将产生式一序号的形式体现，wh 和 el 为 while 和 else 的缩写。关于输出的代码参见 ParsingTable 的 printTableMap 方法。

3.2 语法分析结果展示

语法分析的测试分为两部分：正确代码测试和错误代码测试。测试程序分别为 doc/test 文件夹的 input1.txt 和 input2.txt 文件，截图如下。

input1.txt：(正确)

```
while(a==0){
    if(3==e||a==2){
        a=3*(2+3)
    }
    else{
        e=6*2
    }
}
```

input2.txt：(错误，第三行将)写成})

```
while(a==0){
    if(2==2&&a==2){
        a=3*(1}
    }
    else{
        e=6*2
    }
}
```

首先词法分析程序分析代码，得到 tokens1.txt 和 tokens2.txt 文件，截图如下：

tokens1.txt：

```
while,WHILE,1
(,LC,1
a,ID,1
==,EQ,1
0,NUM,1
),RC,1
{,LM,1
if,IF,2
(,LC,2
3,NUM,2
==,EQ,2
e,ID,2
||,DOR,2
a,ID,2
==,EQ,2
2,NUM,2
),RC,2
{,LM,2
a,ID,3
=,Assign,3
3,NUM,3
*,Mul,3
(,LC,3
2,NUM,3
+,Add,3
3,NUM,3
),RC,3
},RM,4
else,ELSE,5
{,LM,5
e,ID,6
=,Assign,6
6,NUM,6
*,Mul,6
2,NUM,6
},RM,7
},RM,8
```

tokens2.txt：

```
while,WHILE,1
(,LC,1
a,ID,1
==,EQ,1
0,NUM,1
),RC,1
{,LM,1
if,IF,2
(,LC,2
2,NUM,2
==,EQ,2
2,NUM,2
&&,DAND,2
a,ID,2
==,EQ,2
2,NUM,2
),RC,2
{,LM,2
a,ID,3
=,Assign,3
3,NUM,3
*,Mul,3
(,LC,3
1,NUM,3
},RM,3
},RM,4
else,ELSE,5
{,LM,5
e,ID,6
=,Assign,6
6,NUM,6
*,Mul,6
2,NUM,6
},RM,7
},RM,8
```

可以看到，Token 和程序的每个词法单元是一一对应的关系。

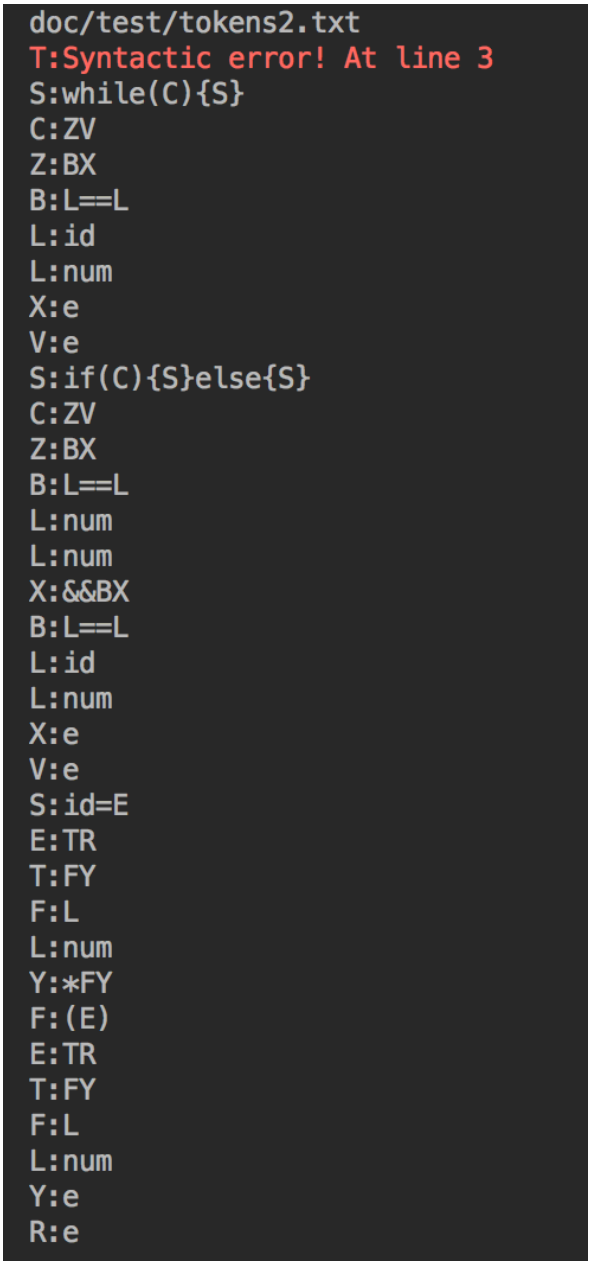
对于 input1 和 input2 的代码，语法分析的结果截图如下：

input1 结果：(在程序中运行 Main 的 main 方法即可得到)

```
doc/test/tokens1.txt
success
S:while(C){S}
C:ZV
Z:BX
B:L==L
L:id
L:num
X:e
V:e
S:if(C){S}else{S}
C:ZV
Z:BX
B:L==L
L:num
L:id
X:e
V:||ZV
Z:BX
B:L==L
L:id
L:num
X:e
V:e
S:id=E
E:TR
T:FY
F:L
L:num
Y:*FY
F:(E)
```

```
E:TR
T:FY
F:L
L:num
Y:e
R:+TR
T:FY
F:L
L:num
Y:e
R:e
Y:e
R:e
S:id=E
E:TR
T:FY
F:L
L:num
Y:*FY
F:L
L:num
Y:e
R:e
```

input2 结果：



如图所示，对于符合文法的程序给出 success 提醒并打印分析序列，对于不符合文法的程序打印出错行数并给出截止到出错行数的分析序列。

4. 总结

在实验中，通过将第一次实验的词法分析程序运用到语法分析程序中，对编译原理的基本思想有了进一步的认识。