## 编译原理PA1-B实验报告

2015011287 方闽杰

本次实验的主要内容依然是在已有的实验框架上，完成词法分析与语法分析。其中，词法分析的部分沿用上一次实验的代码，只需修改、补充Parser.java的代码，以完成语法分析。

步骤一在于理解对于LL（1）文法的解析方式。实验框架中使用了课上描述的自顶向下分析方法，但经过了改进，无需对每一个非终结符都实现parse方法，而是将每个非终结符用一个int型来表示，而后用统一的parse方法就能够通过递归的方式对所有的非终结符进行解析。

步骤二在于实现错误恢复。在这一步骤当中我遇到了比较多的困难。首先是对文档当中错误恢复方法的理解，遇到了不匹配的终结符，该如何报错，报几次错，要如何跳过不匹配的终结符来寻找同步符号，对于不同的同步符号应当如何处理，这些都花了我比较长的时间来弄清楚。在初步实现了错误处理之后，运行编译器又重复地报错，java抛出了NullPointerException的异常。我经过仔细阅读Lexer、Parser和Table中的代码，也试过将错误恢复失败时的返回值改为new SemValue，但都没有解决这个错误。最终，我通过在运行过程中打印大量日志发现，解析失败的返回值必须是null，而Table中的act方法遇到传入的null值时，很有可能会抛出NullPointerException错误。为了保证错误由编译器来提示，而非java来抛出，需要将act的调用用try/catch代码块来包围起来，捕获这一错误。通过这个方式，我实现了文档中提供的错误恢复方案，也避免了java虚拟机报错，并且通过了提供的所有测例。但我并不确定这是最优的方法，有可能会出现其他的错误。

第三个步骤是补充新增的语法特性到parser.spec文件当中。前面复数、case表达式等都比较容易实现，基本与上一次实验的代码相同。但串行循环卫士的部分则有些难度，直接使用文档中的文法会发现Table.java中出现了unreachable code。实际上这是因为我提供的文法并非严格的LL(1)文法。随后，我修改文法如下：

DoStmt ::= do DoSubStmt DoBranchList od

DoBranchList ::= ||| DoSubStmt DoBranchList | ε

该文法符合要求，输入到spec文件当中后，成功地生成了满足测例要求的decaf.jar文件。

思考题：

1. Wiki中有这样的文法说明：

The default setting is unstrict mode. When conflicts appear, we assign higher priority to former defined productions. For instance, the rule definition

S : r1 | r2 | r3

implies that S -> r1 has the higher priority than S -> r2, and S -> r2 has the higher priority than S -> r3。

在parser.spec中，我们使用的文法是ElseClause : ELSE Stmt | ε 。因此，在语法分析时会优先匹配一个新的ELSE分支，找不到时才认为ElseClause为空。

例如以下代码：

1. **class** Main{
2. **static** **void** main() {
3. **int** i;
4. **int** a;
5. **if**(i>0)
6. a = 1;
7. **if**(i<0)
8. a = -1;
9. **else**
10. a = 0;
11. }
12. }

在匹配时，第二个if会根据优先级，先试图与else匹配，并且匹配成功。而后第一个if的ElseClause匹配时，由于没有else可与之匹配，因此选择empty分支。

1. 误报代码如下：
2. **class** Main[
3. **static** **void** main(){
4. **int** a;
5. **int** b;
6. }
7. }

本应只报第一行一处错误，但是编译器输出如下：

\*\*\* Error at (1,11): syntax error

\*\*\* Error at (6,1): syntax error

这是因为，Main之后试图匹配 { ，匹配失败后并未消耗终结符 [ ，会将 [ 作为其他的非终结符的初始字符进行匹配，匹配到最后发现并没有 ] 作为结束，因此再次报错。