1. 程序实现功能

通过antlr对SysY语言书写的源代码进行语法分析：

- 当输入文件存在语法错误时：输出语法错误在哪一行
- 当输入文件不存在语法错误时：按照规定格式输出语法树(含高亮)

2. 如何实现

1. 根据SysY语言定义编写SysYParser.g4和SysYLexer.g4，然后利用antlr生成语法分析器SysYParser.java和词法分析器SysYLexer.java。
2. Main.java接收一个文件路径的参数，然后将文件内容传给词法分析器；再将根据词法生成器生成的CommonTokenStream传给语法分析器。
3. 实现一个继承自BaseErrorListener的ErrorListener并添加给sysYParser，并override BASEErrorListener的syntaxError方法，使语法分析器遇到错误时，打印出错的行号。
4. 通过sysYParser获得SysY语言的语法规则数组ruleNames；通过sysYLexer获得终结符数组symbolicNames。
5. 编写继承自SysYParserBaseVisitor的Visitor类，重写visitChildren和visitTerminal两个函数，然后利用visitor遍历语法树：
    - visitChildren(RuleNode node)：通过node.getRuleContext().depth()-1得到此节点的深度；再通过node.getRuleContext().getRuleIndex()获得语法规则的编号，从ruleNames得到对应的名称并打印；然后再对这个节点每一个子节点进行遍历。

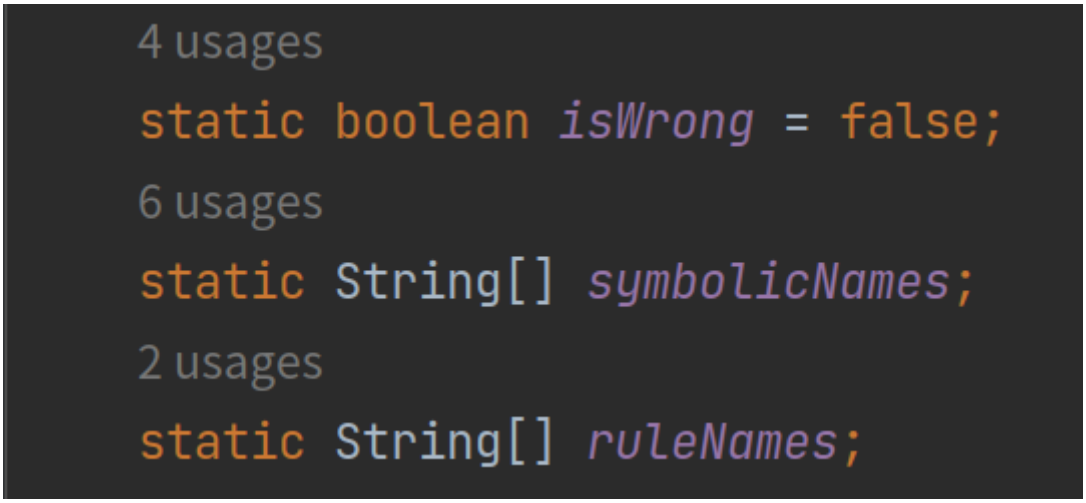
```
// 遍历子节点
for(int i=0; i<node.getChildCount(); i++){
    ParseTree child = node.getChild(i);
    Visitor visitor = new Visitor();
    visitor.visit(child);
}
```

    - visitTerminal(TerminalNode node)：通过node.getParent()得到父节点father，然后通过father.getRuleContext().depth()-1得到父节点的深度，再加上一得到此节点的深度；通过node.getSymbol().getType()-1得到词法规则的编号，然后从symbolicNames中得到对应的名称，并根据规则判断相应的颜色并打印。

3. 精巧设计

3.1 巧用全局变量

我使用了以下三个全局变量：



其中isWrong用于判断代码的语法有没有出错。如果语法分析器遇到错误，isWrong置为true，这样就不会输出正确代码的语法分析树了。ruleNames和symbolicNames分别是语法规则数组和词法规则数组。这样Visitor类就可以根据语法或词法规则下标来获得对应名称。

3.2 如何获得节点深度

一开始我是用一个全局变量depth来存储深度，初始值为0。在visitChildren这一方法中，每次遍历子节点之后depth加一，遍历子节点之后depth减一；不过后来我发现RuleNode可以通过getRuleContext().depth()-1来获得节点深度，但是TerminalNode没有这个方法。于是我先通过node.getParent()得到父节点，并将父节点转化为RuleNode类型，于是获得父节点的深度以后再加上一，就得到了此节点的深度。

3.3 表驱动

通过表驱动方式获得词法规则名称相应的颜色。

4. 印象深刻的bug

4.1 语法规则命名

funcFParams和bType写成了funcfParams和btype，找了很久😅。

4.2八进制和十六机制打印

一开始八进制和十六进制没有转化为十进制打印，因为这次实验要求上没有写；后来想到实验一有这个要求于是转为了十进制打印。