> 实验内容：编写SysY语言的语法分析器，并实现高亮。

实验思路

首先需要根据SysY语言定义编写Parser，这部分基本上就是将手册上的语法规则改写成`Antrl`语句即可。

然后就可以到`Main`类中编写相应的逻辑：

```
public static void main(String[] args) throws IOException {
        if(args.length == 0){
            System.err.println("input path is required");
        }
        // get input file and generate the Lexer
        String source = args[0];
        CharStream input = CharStreams.fromFileName(source);
        SysYLexer sysYLexer = new SysYLexer(input);

        // generate the Parser
        CommonTokenStream tokens = new CommonTokenStream(sysYLexer);
        SysYParser sysYParser = new SysYParser(tokens);

        Visitor visitor = new Visitor();

        // add error listener
        sysYParser.removeErrorListeners();
        MyParserErrorListener myParserErrorListener = new
MyParserErrorListener(visitor);
        sysYParser.addErrorListener(myParserErrorListener);

        // DFS the tree
        ParseTree tree = sysYParser.program();
        visitor.visit(tree);
    }
```

类似于Lab1，需要实现一个自定义的`ErrorListener`，传递给`Parser`，使得在发现语法错误时执行报错输出。

> 为什么要传递`Visitor`？
>
> 由于一旦出现语法错误，就不需要打印语法树了，所以需要在`ErrorListener`监听到语法错误时让`Visitor`不要做输出。

```

```

```
static class MyParserErrorListener extends BaseErrorListener{
    Visitor visitor;

    public MyParserErrorListener(Visitor v){ this.visitor = v; }

    public void syntaxError(Recognizer<?, ?> recognizer,
                            Object offendingSymbol,
                            int line,
                            int charPositionInLine,
                            String msg,
                            RecognitionException e)
    {
        this.visitor.hasError = true;
        System.err.println("Error type B at Line " + line + ": " + msg);
    }
}
```

最后也是最关键的，编写继承自 `SysYParserBaseVisitor<Void>` 的 `Visitor`，实现打印语法树以及高亮的功能。

1. 如何打印节点信息：在 `visitChildren` 和 `visitTerminal` 两个函数中，调用参数 `node` 的相关方法就可以获得 `type`、`text` 等信息。
2. 如何实现缩进：存在 `depth` 字段，调用 `node.getRuleContext().depth()` 获得；注意在 `visitTerminal` 函数中需要强转一下类型。
3. 如何实现高亮：在 `SysYParser` 中找到对应的存放节点类型的数组，将数组中的值修改为对应的颜色值，每次根据 `type` 获取对应颜色。

```
public class Visitor extends SysYParserBaseVisitor<Void>{

    public boolean hasError = false;

    private String getColor(int type) {
        if (type < 0 || type >= _COLOR_NAMES.length) return "";
        return _COLOR_NAMES[type];
    }
    private static final String[] _COLOR_NAMES = {
            null, "[orange]", "[orange]", "[orange]", "[orange]", "[orange]", "
[orange]", "[orange]", "[orange]",
            "[orange]", "[blue]", "[blue]", "[blue]", "[blue]", "[blue]", "
[blue]", "[blue]", "[blue]",
            "[blue]", "[blue]", "[blue]", "[blue]", "[blue]", "[blue]", "[blue]",
"", "", "",
            "", "", "", "", "", "[green]",
            "[red]", "", "", ""
    };
    private void printIndents(int depth){
        for (int i = 0; i < depth * 2; i++){
            System.err.print(" ");
```

```
        }
    }
    @Override
    public Void visitChildren(RuleNode node) {
        int ruleIdx = node.getRuleContext().getRuleIndex();
        String rule = SysYParser.ruleNames[ruleIdx];
        if (!hasError){
            int depth = node.getRuleContext().depth();
            printIndents(depth-1); // need -1: depth begin from 1
            System.err.println(rule.toUpperCase().charAt(0) + rule.substring(1));
        }
        return super.visitChildren(node);
    }
    @Override
    public Void visitTerminal(TerminalNode node) {
        int type = node.getSymbol().getType(); // node type
        String color = getColor(type);
        if (!hasError && !color.equals("")){ // "" means the terminal node we
need, such as '{' '}'
            String literal_name = node.toString();
            String symbol_name = SysYParser.VOCABULARY.getSymbolicName(type);
            // deal with numbers
            if (type == SysYParser.INTEGR_CONST){
                literal_name = convert_to_dec(literal_name);
            }
            int depth = ((RuleNode)node.getParent()).getRuleContext().depth(); //
the depth in parser tree
            printIndents(depth);
            System.err.println(literal_name + " " + symbol_name + color);
        }
        return null;
    }

    private String convert_to_dec(String number){
        if (number.equals("0")){
            number = "0";
        } else if (number.startsWith("0x") || number.startsWith("0X")){
            number = Integer.parseInt(number.substring(2), 16) + "";
        } else if (number.startsWith("0")){
            number = Integer.parseInt(number.substring(1), 8) + "";
        }
        return number;
    }
}
```

碰到的问题

缩进

如何按层次缩进是做实验的时候碰到的最大的一个问题。一开始没想到会有`depth`字段，一直在想如何通过一个变量来标记层次，结果一直没有成功，因为两个`visit`函数都是进入节点之前调用，而没有对应的离开后调用的函数，所以不能通过在两个函数内的增减变化实现层次的变化（也许`Listener`应该是可以的）

拼写

`INTEGR_CONST`。测试的时候看它输出在文件里报拼写错误，还以为是自己写错了，全部改成`INTEGER_CONST`，结果`OJ`过不了，后来发现在`Lab1`的`Lexer`中写的就是`INTEGR_CONST`。