H1 编译原理实验一报告

181860109 **吴润泽** 181860109@smail.nju.edu.cn

H2 实现功能

H3 基本功能

H4 词法分析

词法分析实现借助工具 GNU Flex，编写该词法单元对应的正则表达式和匹配的动作，来生成扫描器。

H5 终结符结点

当匹配了一个合法词法单元后，扫描器根据词法单元的名称和内容创建一个终结符结点，以便于语法树的构建。

H5 非法词法单元

对于未定义的字符或者不合法的数字串，我进行了相应的匹配和报错。

我的额外任务并非要求实现匹配八进制、十六进制、指数，但是为了减轻语法分析错误恢复的负担，我在词法分析中将其对应的合法形式都进行了匹配，建立对应的结点，并进行报错。

H4 语法分析

语法分析实现借助工具 GNU Bison，编写文法对应的生成式和对应的动作，来生成语法分析器。

H5 语法树（实验亮点）

本次实验中，语法树采用**二叉树**的结构来实现多叉树，即每个结点记录其第一个子结点和其第一个兄弟结点。当匹配了一个合法生成式后，生成式的右值中的首个非空项作为左值（非终结符）的子结点，其余项依次作为上一项的兄弟结点。采用先序遍历的方法即可正确打印语法树。

这种实现方式，大大简化了语法树构建的难度，便于实现封装良好的结点构造函数，减少构造结点时的代码冗余，逻辑也更加清晰。

错误恢复（实验亮点）

错误恢复采用**自顶向下**的匹配方式进行考虑。对于 $FunDec, ParamDec$ 等涉及范围较小的则交给上层进行错误匹配，以尽可能避免错误地移入其它合法语句块的情况；对于 $Stmt$ 这种存在语句块嵌套的文法，则尽可能细致地去匹配。另一方面，考虑到 Exp 可能出现的错误种类很多，在其对应文法中也进行了错误恢复的处理。

自顶向下的考虑方式，一方面使得自己不必陷于思考较小的子文法的琐碎细节和可能的错误类型；另一方面，在含有较多语句块的文法进行 $error$ 的匹配，从而使得错误恢复能较快地结束，尽可能少的影响较大的语句块。经过自己的测试，实现出的错误恢复效果还不错，但是仍然存在一些诸如括号不匹配带来的误弹栈的情况。

额外功能

识别注释

本实验所分配的额外任务为**要求1.3**，识别 `//` 和 `/*...*/` 形式的注释，将其滤除。本要求全部在**词法分析**中实现。若文件中出现 `/*` 或 `*/`不匹配的情况，会报词法类型错误。

编译&测试方式

进入 `Code` 文件夹所在路径，执行 `make` 命令，即可获得 **parser** 可执行文件。

执行 `./parser filename` 命令，即可对一个待测试的源文件进行词法和语法的分析。

实验感悟

本次实验是编写一个编译器的开始，实验的难点在于语法树框架的搭建以及如何尽可能编写合理的错误恢复，实验大部分时间都花在了错误恢复的修改和调试上。总体来说，实验完成的还算顺利，加深了对于词法分析中的有限状态自动机和语法分析中LALR文法的理解。