# 编译原理实验报告

221240073 李恒济*

2025 年 3 月 16 日

## 1 完成进度

完成了所有的必做和选做任务，生成了一个命令行工具 parser 可对 $C--$ 程序进行**词法分析**和**语法分析**，可以识别程序中的词法和语法错误并指出相应位置，不存在词法或语法错误时则打印语法分析树.

## 2 编译方式

使用了课程网站上指定的 Makefile，进入 Code/文件夹后键入 **make** 即可在该文件夹下生成 parser.

## 3 重要细节

### 3.1 构造语法树

根据附录 A 给出的正则文法和上下文无关文法编写相应的 Flex 和 Bison 程序，并将词法单元和语法单元视作一个个节点来存储，在进行文法识别的时候自底向上地维护树的结构,识别成功则从根节点开始深度优先的按照要求打印语法树. 值得一提的时，在构造语法树的时候，使用 变长参数，极大提高了编程的便利性.

*Email:221240073@smail.nju.edu.cn

## 3.2 消灭注释

//形式的注释相对容易处理，而/*....*/类型的的注释则相对难处理一些. 考虑到其本质上是找/* 后的**第一个** */，我们从/* 开始不停的匹配非 * 字符或/* 直到找到一个 */结尾即可.

## 3.3 错误处理和恢复

这是我个人感觉最难处理的部分. 手册要求我们要能找出所有的语法错误和词法错误，对于词法错误还好，我们容易编写出识别错误或非法字符的正则表达式. 而对于语法分析部分，如何在 Bison 代码中插入 error 需要仔细思考. 通过查阅文献、询问 GPT 和不断尝试，我采取以下原则来插入 error：

- 高层级规则优先，提高恢复成功率.
- 分隔符处恢复, 在 ; , } 等分隔符处放置 error，减少丢弃的输入量.
- 避免过低级别的错误恢复, 例如 Exp 这样的低级别规则通常不适合作为错误恢复点.

此外，还应注意到，在文法中插入 error 可能会导致冲突，冲突将导致分析器的行为难以理解，因此在插入 error 时应尽量避免冲突.