实验一：词法分析与语法分析 实验报告

成员	学号	院系	邮箱
张路远	221220144	计算机学院	craly199@foxmail.com
姜帅	221220115	计算机学院	jsissosix1221@gmail.com

代码结构

```
Code
├── Makefile
├── lexical.l
├── syntax.y
├── main.c
├── tree.h
└── tree.c
```

`Makefile`：编译脚本文件

`lexical.l`：词法分析器源文件， 描述 token 的正则表达式规则

`syntax.y`：语法分析器源文件，定义语法规则和解析动作

`main.c`：主程序文件

`tree.h & tree.c`：语法树相关定义及实现

功能实现

语法树

- 数据结构设计
  - 语法树的实现基于 `Node` 结构体，定义在 `tree.h` 中，成员包括了节点的名称，行号，类型，子节点数量，存储节点的具体值等
  - 采用链表形式表示，子节点通过 `first_child` 和 `next_sibling` 连接，便于动态扩展和遍历
  - 节点类型用一个枚举类型`NODE_TYPE type`表示
- 功能实现

  语法树的操作函数定义如下：

```
Node* createNode(char *name, NODE_TYPE state, int yylineno, int
child_num, ...);
bool CHECK_ALL_NULL(Node* node);
void print_syntax_tree(Node* node, int index);
```

`createNode`动态创建语法树节点并连接指定数量的子节点，初始化节点字段。通过可变参数列表`va_list`接收子节点指针

`CHECK_ALL_NULL`递归检查节点及其子树是否全部为空节点类型(`NODE_TYPE_NULL`)，在打印语法树时帮助识别无意义的子树

`print_syntax_tree`结合 `CHECK_ALL_NULL` 判断，若子树非全 `NULL`，根据节点类型打印对应信息，并递归处理子节点；若全 `NULL`，将节点标记为 `NODE_TYPE_NULL`。

词法分析

- 错误记录
  - 使用全局变量`Aerrors`来统计词法错误的数目；发现词法错误时更新数目
  - 主程序`main.c`依赖此计数决定是否打印语法树
- 词法分析
  - 根据实验手册提示跟踪行号列号，从而允许在输出时能提供行号
  - 主要解析参考手册附录定义
- 选做(组号3)：注释
  - 发现单行注释`//`时，利用`//.*`来匹配整行，并不做任何动作，达到自动跳过的效果
  - 发现注释起始符`/*`时，维护`pre`、`next`一前一后两个探针，根据探针情况来判别注释结束符`*/`（旧设计为单探针+状态转移，复杂且不直观）

语法分析

- 错误记录
  - 使用全局变量`Berrors`来统计语法错误的数目；发现词法错误时更新数目
  - 主程序`main.c`依赖此计数决定是否打印语法树
- 默认属性
  - 将默认属性均改为适配语法树的数据结构`Node`，在`Node`内部储存属性、信息等
- 悬空else
  - 根据实验手册提示，定义了`LOWER_THAN_ELSE`来解决悬空else问题
- 词法分析
  - 根据实验手册附录先搭建不带语义动作和`error`错误处理的“原始版”；然后添加语法动作，边解析边构造语法树；最后扩展测试集，根据测试结果并结合手册指导插入`error`
  - 主要解析参考手册附录中的文法定义

编译方法

在`Code`文件夹内执行`make`即可，输出的`parser`在同一文件夹内

在根目录执行`./Code/parser ./Test/test.cmm`即可测试单个输入文件