编译原理实验 1 报告

姓名：黄奕诚　　**学号**：161220049

1　开发环境

- 操作系统：macOS Mojave 10.14.3
- 编辑器：Visual Studio Code 1.32.1
- C 编译器：Apple LLVM version 10.0.0 (clang-1000.10.44.4)
- Flex 版本：flex 2.5.35 Apple(flex-31)
- Bison 版本：bison (GNU Bison) 2.3

注：开发完成后已部署到 Ubuntu 18.04 上进行再测试

2　功能介绍

- 完成基本要求
- 完成八（十六）进制数、科学记数法、两种注释风格这三个选做要求，并能判定相应的词法/语法错误

3　编译运行方法

- 编译:在 Code/目录下运行 $ make 即可,随后在项目根目录生成可执行文件 parser. **由于我在 macOS 上进行开发，将 Makefile 中编译选项-lfl 改为-ll，如果编译遇到问题不妨检查一下这里.**
- 运行：在项目根目录执行 $ ./parser [测试文件名] 即可

4　实现中重点细节

4.1　词法分析

- 实现很简单，实验讲义讲得很清楚，用 flex 写起来基本没有障碍.

- 最有挑战性的应该是/* ... */形式的合法注释的正则表达式编写. 首先，如果不用正则表达式，可以手动 input() 的方法找到与/* 对应的最先的 */，这很简单，但为了代码的优雅性，我和黄毅飞、赵士轩两位同学讨论了匹配与/* 最近的 */的正则表达式的方法，并参考了网站 http://www.cs.man.ac.uk/~pjj/cs212/ex2_str_comm.html的思路，分析了状态机生成正则表达式的过程，得到可以从理论上证明正确性的答案（在实验报告里写这个表达式的时候，就想换 Markdown 了）:

  $$“/*”([\wedge*]*“*”+[\wedge*/])*[\wedge*]*“*”+“/”$$

  我觉得真要认真检查注释识别的正确性的话，大概可以 hack 掉很多人（逃

- 其次是八进制数的词法错误检查. 我觉得这一点很难精确认定，比如说 01E4，如果讲义里没有明确要求科学记数法前面浮点数必须要有点号，那么它可能是

  1. 正确的浮点数
  2. 错误的八进制数（E 的出现）
  3. 错误的十六进制（漏写了一个 x 或 X）

  当然，在明确了科学记数法的要求后，偏向于认定其为错误的八进制数. 我想表达的即是

  1. 若明确要求要检出八进制、十六进制数的词法错误，应当给出更为明确的条件
  2. 若判定条件不明确，不妨完全不识别错误的八进制、十六进制数，将问题留给语法，判定为语法错误

4.2 语法分析

- 首先，YYERROR_VERBOSE 和 YYDEBUG 是真的好用.
- 整个过程就是抄语法手册，当搬砖工人 + 排版工人.
- 对于语法错误的判定，我主要针对如讲义里所说的分号、大中小括号附近的错误，并根据测试样例进行了试探性修改. 这边的需求是“错误类型及行号正确即可”，错误类型固然没问题，剩下的问题就是行号，也就是说检出一个语法错误必须要和测试代码认为的行号一致. 这就带来一个问题：什么叫做**该行出现了语法错误**？诚然，绝大多数语法错误我们可以直接肉眼识别是出现在哪一行，并且讲义中某些样例也允许出现多种判定情况（都认为正确），但我认为严谨地定义**“某行有语法错误”**是必要的，否则在交作业之前也无法切实证明自己写的语法错误检出是正确的.
- 对于空文件（或徒有正确注释）的输入，我直接打印了：Empty file!

4.3 抽象语法树建构

- 将所有的终结符和非终结符的类型设为自己定义的 AST_Node*，这样维护起来特别方便.

- 借用了实验讲解 PPT 的思路，维护了每个结点的 first_child（第一个孩子结点）和 sibling（下一个兄弟结点），这样打印语法树、添加子结点比较方便，也不用维护二维数组.
- 在添加子结点的实现中使用了 C 语言的可变长参数，这样可以简化对子结点添加的调用.

5 建议与吐槽

- 我觉得实验应当允许手写词法分析或语法分析器，不用强制要求使用工具，反正规定生成一个 parser 拿来测试就行，大不了规定手写词法语法分析不额外加分就行了，防止大家全放弃 flex、bison（逃
- 希望参照上一部分提到的几个不明确的点，进行一些改进.
- 我觉得这个实验就像是只能 submit 一次的 OJ，样例比较简单，但提交之后如果 AC 不了，既不能改又会被扣分，就挺蛋疼的.
- 对于作弊查重方面，其实我有一个小建议，就是**允许极少部分引用网上的开源代码，但必须注明出处**.
  - 核心思想：**代码引用识别自动化**
  - 引用代码的部分，可以明确规定一种引用格式，比如代码段前后插入有特殊标记的注释，查重时将此段代码略去即可（但对于可执行代码级别的查重无效）；或是规定一个与引用记数有关的全局变量，在引用代码段前标记为 1，其后标记为 0，这样对于可执行代码级别的查重可能也有效
  - 引用代码的部分必须不能超过总代码量的一定比例（如 1%，可以用行数或指令数来衡量）
  - 引用代码无论只是借鉴原代码的思路，还是照抄，只要进行引用标记，都是合理的.（**主要是有时候即使是看懂了原代码的思路，拿来借鉴，也很难写成编译之后不一样的代码，比如只有一两行的代码，或一个正则表达式**）

  此外，受到计算机系统基础实验和操作系统实验的启发，助教不妨在 Makefile 里面写个 make 时自动 git commit，然后提交时会同时提交所有 git log 的记录，用 git log 来进行作弊查重，也是很有价值的.