https://github.com/lazyparser/becoming-a-compiler-engineer

方舟·编译技术入门与实战

第三课：实验起步

吴伟（@lazyparser）

2019-12-15

PLCT Group, part of ISRC, ISCAS & HelloGCC Community & HelloLLVM Community

错过直播没关系，所有直播都有回看

- 课程配套代码及幻灯片地址（也是提问的地方）
  - https://github.com/lazyparser/becoming-a-compiler-engineer
  - https://github.com/lazyparser/becoming-a-compiler-engineer-codes
- 课程视频回看（包含所有直播及录播视频）
  - https://space.bilibili.com/296494084
- 课程直播地址（可以弹幕或评论区互动）
  - https://live.bilibili.com/10339607

本次课程将介绍

- 如何从零开始创建一个编译器项目
- 一些非常细微的编程习惯
- 一些非常细微的 git 使用习惯
- 一些非常细微的测试和调试的思想

以上内容不是按照所列次序讲解，而是一边敲代码一边介绍

第二课的课后练习（PL/0部分）

- 阅读 PL/0 的语言规范，尝试自己手写几个例子
- 用之前小节提到的可视化工具尝试用RE实现词法分析的匹配部分
- 从GitHub或者gitee上找一些参考的例子，看一看
- 预习 flex 这个工具的语法及使用，跑个例子

重申：重在自学，本课程侧重网上没有的内容

- 包括USTC张昱老师的课程中也有公开PL/0的资料
  - http://staff.ustc.edu.cn/~yuzhang/compiler/2012s/lectures/basic_project.pdf
- 有不少学生或爱好者公开了自己的笔记和代码，可以借鉴
  - https://github.com/gdut-yy/PL0
  - http://jcf94.com/download/2016-02-21-pl0-From_PL0_To_Flex.pdf
  - https://github.com/IsaacZhu/PL0/blob/master/doc/

在 GitHub 上搜 PL0 有大量不同版本的代码可以阅读和参考

词法、语法分析的实现方式之分

- 用 lex/yacc 或类似的生成器来构建

- 手写的语法分析器，以及（或）手写的词法分析器

词法、语法分析的实现方式之分

- 用 lex/yacc 或类似的生成器来构建
  - 可以快速构建原型系统或小语言；非常快速，熟练的一天就能出demo
  - 工业级编程系统也有使用
  - 主要问题是系统规模大了之后，改起来很头疼
- 手写的语法分析器，以及（或）手写的词法分析器
  - 很多工业级编译器都倾向于手写：清晰、速度
  - 开发周期长一些；另外，你以为后续改动就容易驾驭了么☺

开始动手写点代码吧 ☺ …… 先从哪里开始？

开始动手写点代码吧 ☺ …… 先从哪里开始？

- 原则：估计生存期超过24小时的代码，就应该用git管理起来

开始动手写点代码吧 ☺ …… 先从哪里开始？

- 原则：估计生存期超过24小时的代码，就应该用git管理起来
- 习惯：git 的姓名和邮件设置要设置好，个人项目跟公司项目分开

开始动手写点代码吧 ☺ …… 先从哪里开始？

- 原则：估计生存期超过24小时的代码，就应该用git管理起来
- 习惯：git 的姓名和邮件设置要设置好，个人项目跟公司项目分开
- 技巧：设置一些常用的 git 缩写，提高效率

开始动手写点代码吧 ☺ …… 先从哪里开始？

- 原则：估计生存期超过24小时的代码，就应该用git管理起来
- 习惯：git 的姓名和邮件设置要设置好，个人项目跟公司项目分开
- 技巧：设置一些常用的 git 缩写，提高效率
- 技巧：灵活运用 git 的 staged 和 unstaged 两种状态
  - https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

开始动手写点代码吧 ☺ …… 先从哪里开始？

- 原则：估计生存期超过24小时的代码，就应该用git管理起来
- 习惯：git 的姓名和邮件设置要设置好，个人项目跟公司项目分开
- 技巧：设置一些常用的 git 缩写，提高效率
- 技巧：灵活运用 git 的 staged 和 unstaged 两种状态
  - https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository
- push 之前一定要检查：只要没 push，commits 都可以修改

一些编程能力之外的必知必会

- 复制粘贴的时候要注意版权和开源许可证的兼容性

一些编程能力之外的必知必会

- 复制粘贴的时候要注意版权和开源许可证的兼容性
  - 这点我愿意多强调一些：国内目前关注的还太少

一些编程能力之外的必知必会

- 复制粘贴的时候要注意版权和开源许可证的兼容性
  - 这点我愿意多强调一些：国内目前关注的还太少
- 将工具生成的代码，跟你自己的改动，最好能够在 git 上区分开
  - 这样可以方便事后代码评审，并屏蔽掉工具升级带来的影响
  - 如果不进行手工修改，不要跟踪工具生成的代码

一些编程能力之外的必知必会

- 复制粘贴的时候要注意版权和开源许可证的兼容性
  - 这点我愿意多强调一些：国内目前关注的还太少
- 将工具生成的代码，跟你自己的改动，最好能够在 git 上区分开
  - 这样可以方便事后代码评审，并屏蔽掉工具升级带来的影响
  - 如果不进行手工修改，不要跟踪工具生成的代码
- 能够自动化的过程，都应该自动化（今天是 Makefile）

更快更强：如何高生产力的完成工作

- 在写代码的时候就要有所考虑，如何进行测试？
  - 打印一些log依然是主流和有用的调试方式
  - 每一次打印的log都要有意义，是长期保存的代码/系统的一部分，
  - 避免使用「111、test、log」这样的输出信息，用更有信息的文本代替

更快更强：如何高生产力的完成工作

- 在写代码的时候就要有所考虑：如何进行测试？
  - 打印一些log依然是主流和有用的调试方式
  - 每一次打印的log都要有意义，是长期保存的代码/系统的一部分，
  - 避免使用「111、test、log」这样的输出信息，用更有信息的文本代替
- 如何避免调试（Debugging）？结构清晰、测试充分、定位容易
- 如何高效率调试？后续课程再说

方舟·编译技术入门与实战

本次主要是技巧和习惯的养成练习

记得完成上次课程的作业 ☺

https://github.com/lazyparser/becoming-a-compiler-engineer

PLCT Group, part of ISRC, ISCAS & HelloGCC Community & HelloLLVM Community