编译原理期末项目要求分析

要求分析(AI)

该期末项目要求以最多 4 人团队形式，构建一个 “编译器的编译器”，需实现词法 / 语法分析器自动生成及指定语言中间代码生成，通过 Canvas 提交多类材料并参与现场答辩，最终按功能实现、展示等多维度评分。

一、核心目标与能力要求

- 掌握编译器自动构造的经典算法，理解编译原理核心技术与编程方法。
- 提升系统开发全流程能力，涵盖需求分析、算法设计、编码及文档编写。
- 锻炼文献研究、开源资源利用、团队协作、表达展示及批判性思维。

二、核心任务要求

1. **技术选型**：从指定经典算法中选择实现，词法分析可选汤普森构造法等，语法分析可选 LL (1) 或 LR 分析法，中间代码生成需基于属性文法或语法制导翻译。
2. **功能实现**：编译器的编译器需接收词法规则（如正则文法）和语法规则（如上下文无关文法），自动生成对应语言的词法 / 语法分析器；至少支持一种测试语言的中间代码生成，生成的编译器需能处理源程序、输出三地址码等中间代码，并对错误代码报错。
3. **开发规范**：代码需托管在 GitHub 或 Gitee，通过提交记录等体现团队协作与个人贡献；中间代码生成需与语法分析同步进行，采用语法制导翻译并保存到文件。

三、团队与提交要求

- 团队规模：最多 4 人，项目权重占总成绩 30%。
- 提交材料：包括源代码（压缩为 ZIP）、程序设计文档（Word/PDF）、答辩 PPT，需通过 Canvas 提交（每队仅需一次提交）。
- 开源要求：代码需上传至 GitHub/Gitee，README 需包含项目介绍、测试用例设计及测试结果，测试需覆盖 “编译器的编译器本身” 和 “生成编译器的功能” 两类场景。

四、文档与 PPT 规范

程序设计文档

- 核心内容：系统设计（概述 + 详细设计）、系统实现（功能描述 + 关键代码截图）、系统测试（用例设计 + 结果，需保证完整性和挑战性）、AI 辅助工具使用说明。
- 格式要求：逻辑清晰、详略得当、格式专业。

答辩 PPT

- 核心内容：系统功能概述、测试用例介绍、团队协作展示（GitHub/Gitee 相关记录）、系统演示。

五、展示与评分规则

展示安排

- 时间：第 16 周，按团队编号顺序进行；展示 10 分钟（全员必须参加），问答 5 分钟。
- 注意事项：提前在教室测试电脑，确保 PPT 和程序正常运行；展示前需完成 Canvas 材料提交。

评分标准（总分 100%）

- 功能实现（正确性、完整性、创新性等）：50%
- 展示质量：20%
- 文档质量：10%
- 代码仓库公开情况：10%
- 团队协作：10%

六、参考资源

- 编译器工具参考：ANTLR、BISON (YACC)、FLEX (LEX)。
- 编程语言定义参考：PL/0、Decaf。
- 规则格式：词法规则常用正则表达式，语法规则常用乔姆斯基上下文无关文法或巴科斯 - 诺尔范式（BNF）。

具体实现(暂定，待商讨)

语言选择：Python

分工：
张昊天--词法分析部分代码+程序设计文档编写
夏弘泰--语法分析部分代码+代码汇总+答辩PPT制作
陈帆--中间代码生成部分代码+测试案例设计+答辩汇报(不确定，答辩说是全员参加，但不知道汇报要几个人讲)

参考项目结构(AI)：

```
compiler_project/
├── config/                  # 配置文件目录
│   ├── lex_rules.lex        # 词法规则配置
│   └── yacc_rules.bnf       # 语法规则配置
├── src/                     # 源代码目录
│   ├── lexer_generator/     # 词法分析器生成模块
│   │   ├── __init__.py      # 模块入口
│   │   ├── regex_parser.py  # 正则表达式解析（生成NFA）
│   │   ├── nfa_to_dfa.py    # NFA转DFA
│   │   └── dfa_minimizer.py # DFA最小化
│   ├── parser_generator/    # 语法分析器生成模块
│   │   ├── __init__.py      # 模块入口
│   │   ├── grammar_analyzer.py # 语法规则分析（消除左递归、提取左因子）
│   │   ├── first_follow.py  # 计算FIRST集和FOLLOW集
│   │   ├── ll1_table.py     # 构建LL(1)分析表
│   │   └── ast_builder.py   # 基于LL(1)分析表生成AST（抽象语法树）
│   ├── code_generator/      # 中间代码生成模块
```

```
|   |   ├── __init__.py      # 模块入口
|   |   ├── semantic_analyzer.py # 语义分析（类型检查、变量作用域）
|   |   └── ir_generator.py  # 三地址码生成
|   ├── utils/               # 工具函数目录
|   |   ├── __init__.py      # 模块入口
|   |   ├── file_handler.py  # 文件读写工具
|   |   ├── error_handler.py # 错误处理（语法错误提示、位置定位）
|   |   └── visualizer.py    # 可视化工具（状态机图、AST图）
|   └── main.py              # 项目主入口（命令行交互、流程调度）
├── test/                    # 测试用例目录
|   ├── valid_cases/         # 合法测试用例
|   |   ├── test1.txt        # 测试表达式计算
|   |   └── test2.txt        # 测试条件语句
|   └── invalid_cases/       # 非法测试用例（用于错误处理测试）
|       ├── test3.txt        # 语法错误测试
|       └── test4.txt        # 语义错误测试
├── docs/                    # 文档目录
|   ├── design_doc.pdf       # 程序设计文档
|   ├── user_manual.md       # 用户手册（如何使用生成的编译器）
|   └── reference/           # 参考资料（算法原理、开源工具文档）
├── .gitignore               # Git忽略文件（如编译产物、IDE配置）
├── README.md                # 项目说明（团队信息、功能介绍、运行步骤）
└── requirements.txt         # Python依赖库列表（如graphviz、pyparsing）
```

规定：数据格式等见api_spec.md文件