**软件工程实验三实验报告**

**曾鹏飞**

**221830013**

1. **项目内容**

本次实验需要根据实验二所设计的 UML 图，将设计的软件实现为代码，实验一中我选 择的软件是 ToDoList，现将 UML 类图和活动粘贴如下:

图示

描述已自动生成图示

描述已自动生成

1. **实验过程**

我选择了Python语言，并利用ChatGPT大模型辅助开发。同时，我使用GitHub远程仓库进行代码管理。首先，我将实验二中的UML类图源代码输入大模型，并附加了“分多个文件完成”的提示词，以便它能够根据UML类图生成第一版的C++代码。大模型采用了面向对象编程的方法，这与实际应用开发场景相符合。尽管大模型的思路与我一致，即将UML类图中的每个类分别实现并进行耦合，但它生成的代码仅限于框架层面，具体的详细实现并未达到预期（例如，类中每个函数的实现仅有寥寥数行）。

因此，我将大模型生成的代码逐个文件进行细化，这样不仅增加了代码行数，而且更充分地实现了所需的功能。为了进行测试，我需要一个main函数来将所有文件联系起来。我首先创建了一个简单的非交互性main函数，通过调用用户测试模块来验证模块间的编译是否正常。然而，由于文件是分块输入给大模型的，加上GPT 40的每日限额已满，后续生成的代码间的联系变得不那么紧密（例如，在A文件中调用了B文件中不存在的函数）。这需要我继续输入给GPT进行优化，并自行调整。

经过不断的调整和GPT的更改，所有函数最终都能正常编译并运行。接下来，我选择使用Kimi大模型，根据之前用于测试的main函数作为接口，生成一个交互性终端。在这个过程中，如果不注意接口的问题，GPT可能会生成许多“自以为存在”的接口。因此，在prompt中必须明确提示要使用测试main函数的接口，以生成令人满意的结果。

最后，我使用GitHub来记录我最后几次的调整记录。 (遗憾的是，一开始调试各个模块联系的时候忘记git记录了qwq):

网址如下: [moonmoonbirdfly/Software-Engineering](https://github.com/moonmoonbirdfly/Software-Engineering)

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

在代码审查过程中，根据实验要求,我对某些部分使用了pylint的调整建议。pylint是一个代码质量检查工具，它能够对代码进行分析并提出改进建议。尽管我已经根据pylint的建议，对大部分代码进行了优化，但是以下内容:

函数定义（pylint建议函数参数不超过五个，我有少数函数参数达到了六个，但这些函数已经实现完毕，再次修改将会带来不便）以及函数用途的注释说明（即function or method docstring,每个函数功能都很明显,没必要一个一个添加,机械劳动而已），我并没有完全地修改。

但我还是想要强调一点：pylint的检查有时也会对if-else语句的冗余性提出质疑。例如，它可能会建议我删除某个else分支，但如果这样做，将会破坏原有的逻辑结构。因此，我认为pylint的优化建议并不总是完全可靠，我们需要根据实际情况进行判断和取舍。

1. **实验结果**

文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成

功能不一一展示,有凑字数嫌疑…

1. **自我感受**

在本次实验中，我们成功地构建了ToDoList软件的框架，并深入探索了如何利用大型语言模型来辅助软件开发，以及如何运用Git来管理本地和远程代码仓库。大型语言模型在开发过程中提供了巨大的支持，它能够快速生成基础框架代码，让我们能够专注于后续的完善和复杂逻辑的编写。然而，实验报告的撰写过程中不允许使用这种辅助工具，这无疑增加了写作的挑战性，也是导致我的报告篇幅较为简短的原因之一。

Git作为一个高效的代码管理工具，通过对比数据库中的撤销和重做操作，我对Git的”undo”,”redo”功能有了更深入的理解。但是，代码提交的频率确实是一个需要仔细考虑的问题。提交频率过低可能会导致代码丢失，而提交频率过高则意味着未来可能需要花费大量时间来逐个执行”redo”操作。

我个人认为，代码提交的频率应该根据项目的需求和团队的工作流程来决定。一个合理的提交频率可以帮助团队成员更好地跟踪项目进度，同时也能够减少因频繁提交而带来的管理负担。此外，定期的代码审查和合并请求（Merge Request）也是保持代码质量和项目进度的有效手段。