中国石油大学（北京）

2024— 2025 学年秋季学期

《C++程序设计》

结课报告

学生姓名：胡林森

学　　号：2023015509

班　　级：数据 23-3 班

成　　绩：

中国石油大学（北京）克拉玛依校区

2024 年　10　月　24　日

目 录

图表目录

一、 选题：个人记账系统

随着经济的发展和生活水平的提高，人们的消费习惯和财务管理意识不断发生变化。个人财务管理需求的增加促使个人记账工具逐渐成为一种热门工具。这些工具不仅帮助用户合理规划开支，监控财富的流动，更帮助其做出明智的经济决策。

有调查结果显示，对于个人消费情况，只有 23%的同学能够合理并清晰地知道自己的消费去向，89.2%的同学希望能清晰了解自己每一个月的消费去向，并能通过上月的消费情况合理规划出下个月的消费预警提示。因此，开发设计一款实用的记账 App 具有一定的市场应用前景。[1]

本次 C++结课报告完成了一个简单实用的个人记账系统，通过清晰的菜单驱动界面和灵活的数据记录方式，使用户能够快速上手并高效地进行个人财务管理。

同时系统设计采用模块化思路，涵盖了数据结构、功能实现和用户交互等多方面，使得代码易于维护和扩展。

二、 需求分析

(1) 选择菜单

i 主菜单展示

- 显示主要的操作选项，包括记账、查询、保存和退出。
- 用户可以输入 1-4 之间的数字来选择不同的操作。

ii 记账菜单展示

- 提供用户选择记录收入、记录支出和返回上一级菜单的功能。
- 用户可以输入 1-3 之间的数字来选择对应的操作。

iii 查询菜单展示

- 提供用户选择统计所有、统计收入、统计支出和返回上一级菜单的功能。
- 用户可以输入 1-4 之间的数字来选择对应的操作。

iv 操作确认

- 在主菜单中，用户选择退出时，需要确认是否真的退出程序。

v 用户输入处理

- 系统需要能够处理用户的输入，并给予适当的反馈。
- 对于无效输入，系统应该能够提示用户重新输入。

(2) 记账功能

i 收入

- 功能描述: 用户可以录入收入信息。
- 输入需求: 用户需提供具体的收入金额和备注信息。
- 输出需求: 系统需确认记录成功，并将其存储。

ii 支出

- 功能描述: 用户可以录入支出信息。
- 输入需求: 用户需提供具体的支出金额和备注信息。
- 输出需求: 系统需确认记录成功，并将其存储。

(3) 查询功能

i 统计所有

- 功能描述: 用户可以查看所有收入和支出记录。
- 实现方法：

- 输出需求: 系统将显示所有记录及其总收入、总支出和净收入。

ii 统计收入

- 功能描述: 用户查找所有录入的收入信息。
- 输出需求: 系统仅展示收入记录及其总金额。

iii 统计支出

- 功能描述: 用户查找所有录入的支出信息。
- 输出需求: 系统仅展示支出记录及其总金额。

(4) 保存功能

- 功能描述: 用户可以将所有账目保存至指定文件。
- 输入需求: 用户需提供文件路径以进行保存。
- 输出需求: 系统需确认保存成功或失败，并处理相应的文件错误。

(5) 退出功能

- 功能目标：用户在主菜单选择退出选项（选项 4），系统需要提示用户确认是否真的想退出程序。
- 用户交互：当用户选择退出后，程序需要询问用户确认以避免误操作。用户可以输入'Y'或'y'来确认退出，输入其他字符则继续留在主菜单。
- 程序终止：如果用户输入'Y'或'y'，调用 exit(0)函数来终止程序。程序应该正常终止，并释放所有资源。
- 资源管理：确保在调用 exit(0)之前，已经处理完所有必要的资源释放和状态保存工作，以避免数据丢失。此功能目前可通过调用保存功能实现。

三、 总体设计

(1) 项目结构

将系统分为三个文件，其中一个为 main.cpp，另外两个为头文件 menu.h 以及 operations.h；main.cpp 源文件主要完成对驱动菜单函数的调用，menu.h 中主要实现菜单打印、功能函数选择，operations.h 中只要对各种操作进行声明与实现。将系统分为这三个板块有利于后期维护与功能拓展。

(2) 数据结构

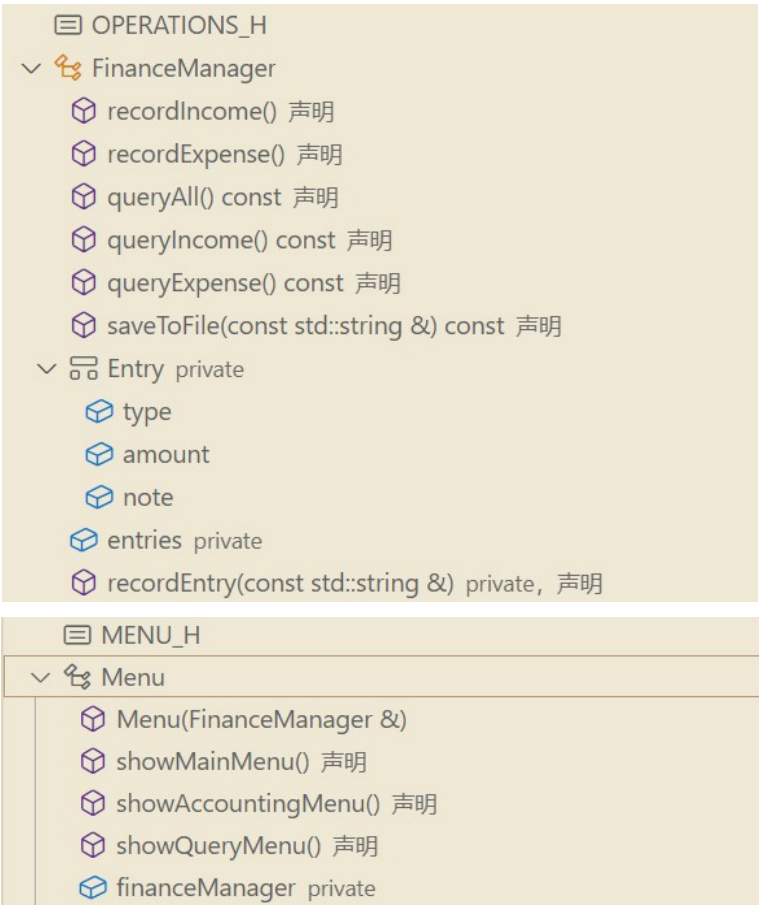
在 operations.h 中设计 FinanceManager 类，封装结构体 Entry 包含三个成员：type（记录类型，如收入或支出 string 型）、amount（金额 float 型）、note（备注 string 型）。

通过这种封装，使得每一条财务记录都可以完整地描述其必要的信息。使用 STL std::vector 动态数组来存储 Entry 对象，允许灵活地添加新记录。对记录数量不确定的财务条目非常有用。

在 menu.h 中设计 Menu 类，定义了一个私有成员 financeManager，它是 FinanceManager 类的引用。这意味着 Menu 类可以通过此成员直接访问 FinanceManager 中定义的功能函数，比如记录收入、支出，以及查询财务数据等。

(3) 函数申明

图表 3.1 各类函数结构设计



四、 详细设计及编码

通过“#ifndef MENU_H　#define MENU_H”类似头文件保护宏，避免头文件被多次包含。

(1) menu.h

```
#ifndef MENU_H
#define MENU_H

#include <iostream>
#include "operations.h"

class Menu {
public:
    Menu(FinanceManager& manager) : financeManager(manager) {}

    void showMainMenu();
    void showAccountingMenu();
    void showQueryMenu();
```

```
private:
    FinanceManager& financeManager; // 引用 FinanceManager 对象
};

void Menu::showMainMenu() {
    int choice;
    std::cout << "==================== 主菜单 ====================\n";
    std::cout << "|----------------1. 记账----------------------|\n";
    std::cout << "|----------------2. 查询----------------------|\n";
    std::cout << "|----------------3. 保存----------------------|\n";
    std::cout << "|----------------4. 退出----------------------|\n";
    std::cout << "================================================\n";
    std::cout << "请输入选择 (1-4): ";
    std::cin >> choice;

    switch (choice) {
        case 1:
            showAccountingMenu();
            break;
        case 2:
            showQueryMenu();
            break;
        case 3:
            financeManager.saveToFile("D:\\C++work\\Demo\\data.txt");
            break;
        case 4:
            char confirm;
            std::cout << "确认退出吗? (Y/N): ";
            std::cin >> confirm;
            if (confirm == 'Y' || confirm == 'y') {
                exit(0);
            }
            break;
        default:
            std::cout << "无效选择，请重新输入。\n";
    }
}

void Menu::showAccountingMenu() {
    int choice;
    std::cout << "==================== 记账菜单 ====================\n";
    std::cout << "|----------------1. 收入----------------------|\n";
    std::cout << "|----------------2. 支出----------------------|\n";
    std::cout << "|----------------3. 返回----------------------|\n";
```

```
    std::cout << "================================================\n";
    std::cout << "请输入选择 (1-3): ";
    std::cin >> choice;

    switch (choice) {
        case 1:
            financeManager.recordIncome();
            break;
        case 2:
            financeManager.recordExpense();
            break;
        case 3:
            return;
        default:
            std::cout << "无效选择，请重新输入。\n";
    }
}
void Menu::showQueryMenu() {
    int choice;
    std::cout << "==================== 查询菜单 ====================\n";
    std::cout << "|------------------1. 统计所有--------------------|\n";
    std::cout << "|------------------2. 统计收入--------------------|\n";
    std::cout << "|------------------3. 统计支出--------------------|\n";
    std::cout << "|------------------4. 返回菜单--------------------|\n";
    std::cout << "================================================\n";
    std::cout << "请输入选择 (1-4): ";
    std::cin >> choice;
    switch (choice) {
        case 1:
            financeManager.queryAll();
            break;
        case 2:
            financeManager.queryIncome();
            break;
        case 3:
            financeManager.queryExpense();
            break;
        case 4:
            return;
        default:
            std::cout << "无效选择，请重新输入。\n";
    }
}
#endif
```

定义了一个构造函数 Menu(FinanceManager& manager)：financeManager(manager) {}来接收对 financeManager 的引用。

通过打印提示信息引导用户输入选择代号，在使用 switch 函数选择调用功能函数。

(2) operations.h

```
#ifndef OPERATIONS_H
#define OPERATIONS_H

#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <limits>

class FinanceManager {
public:
    void recordIncome();
    void recordExpense();
    void queryAll() const;
    void queryIncome() const;
    void queryExpense() const;
    void saveToFile(const std::string& filename) const;

private:
    struct Entry {
        std::string type;
        float amount;
        std::string note;
    };

    std::vector<Entry> entries;

    void recordEntry(const std::string& type);
};
void FinanceManager::recordEntry(const std::string& type) {
    Entry entry;
    entry.type = type;
    std::cout << "请输入" << type << "金额: ";
    while (!(std::cin >> entry.amount) || entry.amount < 0) {
        std::cout << "输入无效，请输入一个有效的金额: ";
        std::cin.clear(); // 清除错误状态
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n'); // 清除输入缓冲
    }
```

```
    std::cin.ignore(); // 清除输入缓冲
    std::cout << "请输入备注: ";
    std::getline(std::cin, entry.note);

    entries.push_back(entry);
    std::cout << "完成记账。\n";
}

void FinanceManager::recordIncome() {
    recordEntry("收入");
}

void FinanceManager::recordExpense() {
    recordEntry("支出");
}

void FinanceManager::queryAll() const {
    float totalIncome = 0, totalExpense = 0;
    std::cout << "所有账目:\n";
    for (const auto& entry : entries) {
        std::cout << entry.type << " | " << entry.amount <<  "\n";
        if (entry.type == "收入") {
            totalIncome += entry.amount;
        } else {
            totalExpense += entry.amount;
        }
    }
    std::cout << "总收入: " << totalIncome << "\n";
    std::cout << "总支出: " << totalExpense << "\n";
    std::cout << "净收入: " << (totalIncome - totalExpense) << "\n";
}

void FinanceManager::queryIncome() const {
    float totalIncome = 0;
    std::cout << "所有收入:\n";
    for (const auto& entry : entries) {
        if (entry.type == "收入") {
            std::cout << entry.amount  << "\n";
            totalIncome += entry.amount;
        }
    }
    std::cout << "总收入: " << totalIncome << "\n";
}
```

```
void FinanceManager::queryExpense() const {
    float totalExpense = 0;
    std::cout << "所有支出:\n";
    for (const auto& entry : entries) {
        if (entry.type == "支出") {
            std::cout << entry.amount  << "\n";
            totalExpense += entry.amount;
        }
    }
    std::cout << "总支出: " << totalExpense << "\n";
}

void FinanceManager::saveToFile(const std::string& filename) const {
    std::ofstream ofs(filename, std::ios::app);
    if (!ofs.is_open()) {
        std::cerr << "无法打开文件 " << filename << " 进行写入。\n";
        return;
    }
    ofs << "类型 金额 备注\n";
    for (const auto& entry : entries) {
        ofs << entry.type << " " << entry.amount << " " << entry.note << "\n";
    }
    ofs.close();
    std::cout << "保存成功。\n";
}

#endif
```

1. recordEntry(const std::string& type)

这个函数用于记录一笔账目。当用户要输入一笔收入时，程序会提示用户输入金额和备注信息，并将其存储在内存中的数据结构中。

- 创建一个 Entry 结构体对象，用于保存账目详情。
- 提示用户输入金额，。使用 std::cin 进行输入。
- 错误处理，通过循环检测用户输入的金额，确保输入有效（非负数）。
- 提示用户输入备注信息，使用 std::getline 获取整行输入。
- 将输入的账目信息存储到 entries 向量中。
- 输出"完成记账"提示。

2. recordIncome()

调用 recordEntry("收入")来记录类型为收入的账目。

3. recordExpense();

调用 recordEntry("支出")来记录类型为支出的账目。

4. queryAll() const

const 关键字表明该函数不会修改类的任何成员变量。后续几个函数也有相应设计。

用于查询所有的记录，包括收入和支出等。它会遍历所有的记录，计算总收入、总支出，并显示每一条记录的详细信息。

5. queryIncome const

同理 const 修饰函数，保护数据安全。

用于查询和显示所有收入的记录。它会遍历已记录的条目，仅输出类型为“收入”的记录和总收入。

6. queryExpense() const

用于查询和显示所有支出的记录。它会输出所有类型为“支出”的记录及其总支出。

7. saveToFile(const std::string& filename) const

以追加模式打开文件，以免丢失已有数据，如果文件无法打开，输出错误信息。

将所有的记录保存到一个指定文件中。保存的格式包含记录类型、金额和备注等信息，以供后续查看。

(3) 主函数

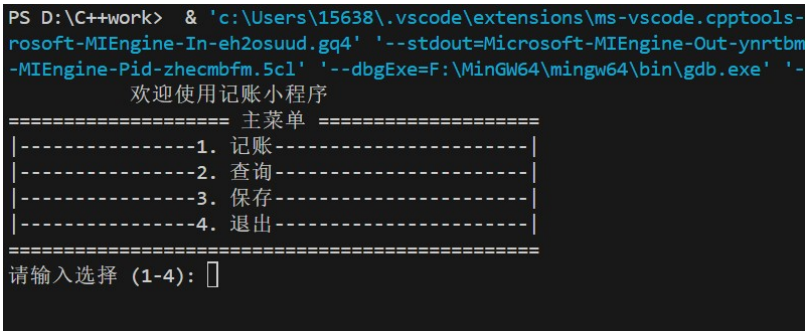
```
#include <iostream>
#include "menu.h"
#include "operations.h"

int main() {
    std::cout << "\t   欢迎使用记账小程序" << std::endl;

    FinanceManager financeManager; // 创建 FinanceManager 实例
    Menu menu(financeManager); // 将 FinanceManager 实例传递给 Menu

    while (true) {
        menu.showMainMenu(); // 调用 Menu 的主菜单方法
    }

    return 0;
}
```

通过 FinanceManager financeManager 以及 Menu menu(financeManager)创建实例，并将其传给 menu。

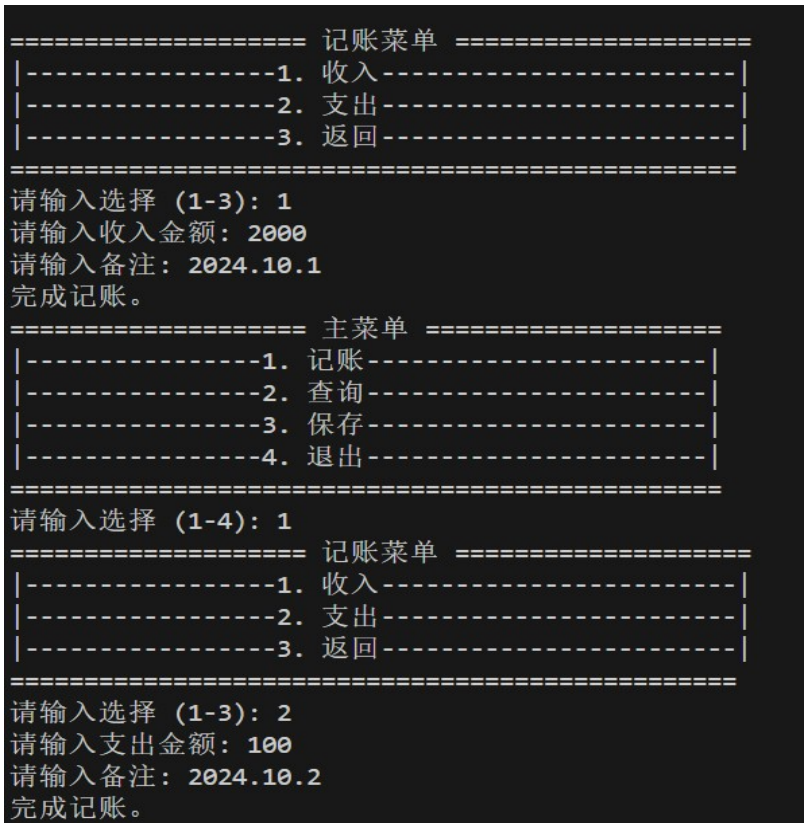
使用 while (true)无限循环 menu.showMainMenu()，直到被调用的函数终止。每次循环时调用 Menu 类的 showMainMenu 方法，显示主菜单的选项供用户选择。

五、 程序运行截图

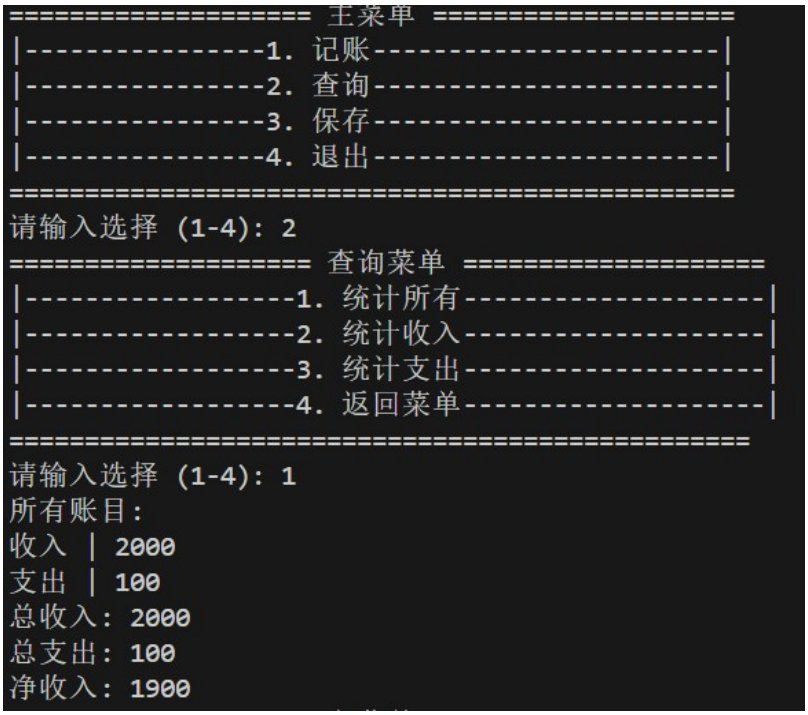
图表 5.1 主菜单



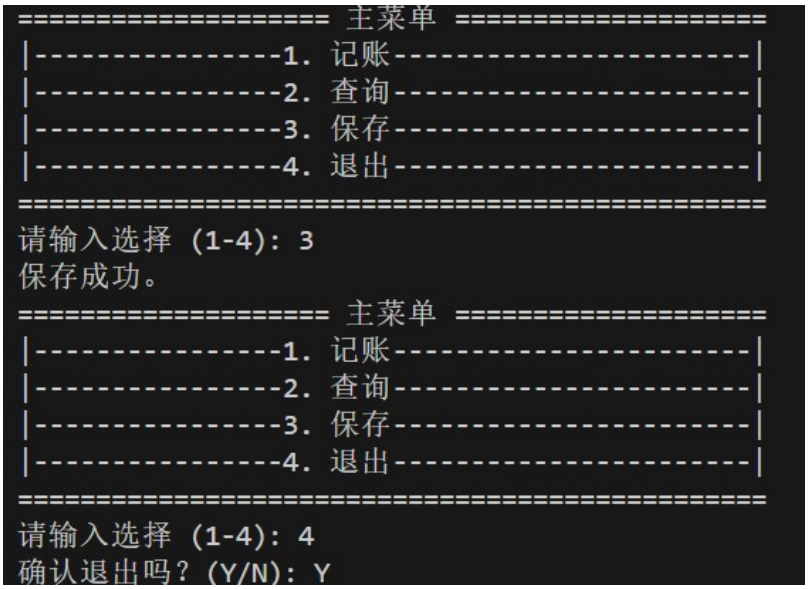
图表 5.2 收入支出记录功能



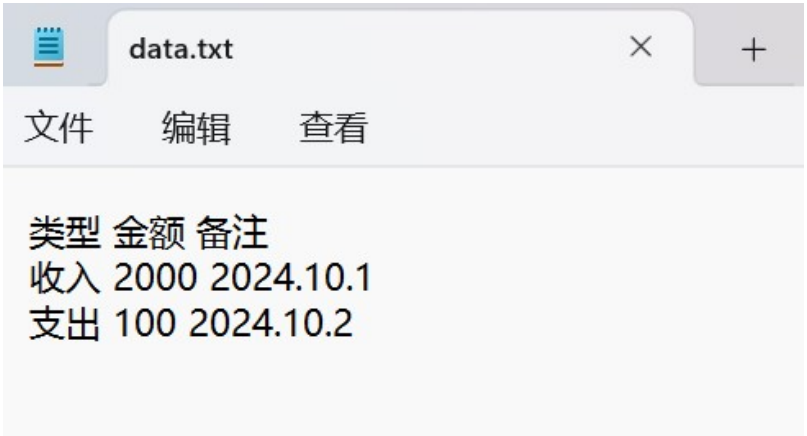
图表 5.3 各类查询功能



图表 5.4 保存与退出



图表 5.5 保存结果



六、 课程学习感受和建议

学习感受：

- C++基本语法，理解数据类型、控制结构以及函数的定义和使用。这些是构建更复杂程序的基础，没有扎实的基本功，很容易在后续的学习中遇到困难
- 面向对象编程（Object-Oriented Programming，简称 OOP），它使用“对象”作为程序的基本构建块。OOP 的核心思想是将对象属性和方法结合到一起，以便更好地模拟和处理现实世界中的事物。以下是 OOP 的几个基本概念：
    - 类：为一个模板，定义了对象的属性和方法；可以通过构造函数完成对象的创建。
    - 对象：是类的实例。对象包含属性和方法可以看作是一个具有状态和行为的实体。
    - 继承：指一个类可以继承另一个类的属性和方法，从而实现代码的重用。子类可以扩展父类的功能，也可以覆盖父类的方法。
    - 多态：多态允许不同类的对象通过相同的接口调用不同的方法。它使得程序更加灵活，便于扩展和维护。多态通常通过方法重载和方法重写实现。
    - 封装：指将对象的属性和方法隐藏在内部，只通过公开的方法与外部进行交互。这种做法保护了对象的内部状态，防止外部代码直接访问和修改对象的属性。

C++ 的模块化、面向对象编程能够在复杂项目管理、代码维护及软件开发效率等方面提供显著的优势。

- 我还认识到学习和实践的重要性，通过反复练习简单的程序，逐渐提升了自己的编程熟练度。

建议：

- 希望加入一些 STL(Standard Template Library)的介绍，比如向量 std::vector、列表 std::list、队列 std::queue、和栈 std::stack 等简单容器

七、 参考文献

[1] 刘露,郦丽,孙雅文.一个在校大学生消费记账 App 的设计[J].电脑知识与技术,2023,19(08):62-64.DOI:10.14004/j.cnki.ckt.2023.0381.

[2] 尚硅谷铁粉. C++编程：小谷记账簿软件项目[EB/OL].(2023-06-14) https://blog.csdn.net/zjjcchina/article/details/131203392