实验报告

题目

实验1：线性表的应用

姓名

叶栩言

学号

2023200033

完成日期

2025年10月9日

---

1. 需求分析

程序设计任务

本实验旨在实现一个任务管理系统，支持用户通过三种不同的数据结构（顺序表、单向链表、双向链表）进行任务的增删改查、排序、保存和加载功能。通过对比三种数据结构的性能，深入理解其特点及适用场景。

输入形式和范围

- **任务名称**：字符串，长度不超过50个字符。
- **任务描述**：字符串，长度不超过200个字符。
- **任务优先级**：整数，范围为1到5，数字越大表示优先级越高。
- **任务截止日期**：日期格式为YYYY-MM-DD。

输出形式

- **任务列表**：包括任务名称、描述、优先级和截止日期。
- **提示信息**：如“任务添加成功”、“任务未找到”等。

程序功能

- **添加任务**：将新任务添加到任务列表。
- **删除任务**：根据任务名称删除任务。
- **修改任务**：更新任务的名称、描述、优先级或截止日期。
- **查询任务**：根据名称或其他条件查找任务。
- **排序任务**：按截止日期或优先级排序任务。
- **保存任务**：将任务列表保存到文件。
- **加载任务**：从文件加载任务列表。

测试数据

正确输入

- 添加任务：名称为“数据结构作业”，描述为“写完王秋月老师布置的Lab”，优先级为5，截止日期为2025-10-15。
- 查询任务：名称为“数据结构作业”。

错误输入

- 添加任务：优先级为6（超出范围）。
- 查询任务：名称为“不存在的任务”。

---

2. 概要设计

抽象数据类型定义

- `Task`：表示任务的数据结构，包含名称、描述、优先级和截止日期。
- `TaskManager`：任务管理器的抽象基类，定义了任务管理的基本操作。

主程序流程

1. 用户选择数据结构（顺序表、单向链表、双向链表）。
2. 进入主菜单，用户选择操作（添加、删除、修改、查询等）。
3. 根据用户输入调用相应的功能模块。
4. 输出操作结果或提示信息。

模块层次关系

- `main.cpp`：主程序入口，提供用户交互界面。
- `SequentialTaskManager`：顺序表实现。
- `LinkedTaskManager`：单向链表实现。
- `DoublyLinkedTaskManager`：双向链表实现。

> 顺序表、单向链表、双向链表是各自独立的数据结构实现，只可选择一种。

---

3. 详细设计

数据类型实现

`Task` 数据结构

```
struct Task {
    std::string name;         // 任务名称
    std::string description; // 任务描述
    int priority;             // 任务优先级（1-5）
    std::string dueDate;      // 任务截止日期（YYYY-MM-DD）
};
```

伪码展示

添加任务

```
输入任务名称、描述、优先级和截止日期
检查优先级是否在范围内
如果合法，创建任务对象并添加到任务列表
输出“任务添加成功”
否则，输出“优先级不合法”
```

删除任务

```
输入任务名称
在任务列表中查找任务
如果找到，删除任务并输出“任务删除成功”
否则，输出“任务未找到”
```

查询任务

```
输入查询条件（如任务名称）
遍历任务列表，查找符合条件的任务
如果找到，输出任务信息
否则，输出“任务未找到”
```

函数调用关系图

```
main.cpp
|
|-- SequentialTaskManager.h
|   |-- SequentialTaskManager.cpp
|
|-- LinkedTaskManager.h
|   |-- LinkedTaskManager.cpp
|
|-- DoublyLinkedTaskManager.h
|   |-- DoublyLinkedTaskManager.cpp
|
|-- TaskManager.h
|-- Task.h
```

---

4. 调试分析

调试问题及解决

- **问题 1**：输入优先级超出范围时程序崩溃。
  - **原因**：未对用户输入的优先级进行有效性检查。
  - **解决**：在输入优先级时增加范围检查，确保优先级在 1 到 5 之间。
  - **代码示例**：

```
if (priority < 1 || priority > 5) {
    std::cout << "优先级不合法，请输入 1 到 5 之间的数字。" <<
std::endl;
    return;
}
```

- **问题 2**：任务名称重复导致数据冲突。
  - **原因**：在添加任务时未检查名称是否重复。
  - **解决**：在添加任务前，遍历任务列表检查是否存在相同名称的任务。
  - **代码示例**：

```
for (const auto& task : tasks) {
    if (task.name == newTask.name) {
        std::cout << "任务名称重复，无法添加。" << std::endl;
        return;
    }
}
```

- **问题 3**：文件保存失败。
  - **原因**：文件路径错误或文件权限不足。
  - **解决**：在保存文件时增加错误处理逻辑，并提示用户检查文件路径和权限。
  - **代码示例**：

```
std::ofstream outFile(filename);
if (!outFile.is_open()) {
    std::cout << "文件保存失败，请检查路径和权限。" << std::endl;
    return false;
}
```

时空复杂度分析

- **添加任务**：
  - 时间复杂度：$O(1)$（顺序表）或 $O(n)$（链表）。
  - 空间复杂度：$O(n)$。
- **删除任务**：
  - 时间复杂度：$O(n)$。

    - 空间复杂度：$O(n)$。
- **查询任务**：
    - 时间复杂度：$O(n)$。
    - 空间复杂度：$O(n)$。

经验和体会

通过调试过程，发现了程序设计中的多个潜在问题，并通过逐步优化提高了代码的健壮性和可维护性。尤其是在处理用户输入和文件操作时，增加了更多的错误处理逻辑，确保程序在各种边界情况下都能正常运行。

---

5. 用户使用说明

操作步骤

1. **编译项目**： 使用以下命令编译项目：

   ```
   g++ -o TaskManager main.cpp SequentialTaskManager.cpp
   LinkedTaskManager.cpp DoublyLinkedTaskManager.cpp -std=c++11
   ```

   如果编译过程中出现错误，请检查是否安装了 GCC 编译器，并确保所有源文件都在同一目录下。

2. **运行程序**： 使用以下命令运行程序：

   ```
   ./TaskManager
   ```

   程序启动后会显示主菜单，用户可以根据提示选择操作。

3. **选择数据结构**： 程序启动时会提示用户选择数据结构（顺序表、单向链表或双向链表）。根据实际需求输入对应的数字（1、2 或 3）。

4. **执行任务管理操作**：

    - **添加任务**：选择菜单中的“添加任务”选项，输入任务名称、描述、优先级和截止日期。
    - **删除任务**：选择菜单中的“删除任务”选项，输入要删除的任务名称。
    - **修改任务**：选择菜单中的“修改任务”选项，输入任务名称和新的任务信息。
    - **查询任务**：选择菜单中的“查询任务”选项，输入查询条件（名称或截止日期）。
    - **排序任务**：选择菜单中的“按截止日期排序”或“按优先级排序”选项。
    - **保存任务到文件**：选择菜单中的“保存任务到文件”选项，输入文件名。
    - **从文件加载任务**：选择菜单中的“从文件加载任务”选项，输入文件路径。

5. **退出程序**： 在主菜单中选择“退出系统”选项，程序会安全退出并释放所有资源。

Tips

- 输入任务名称时，请确保名称唯一，否则会导致添加失败。
- 输入日期时，请使用“YYYY-MM-DD”格式，确保日期合法。

- 文件操作时，请确保文件路径正确且具有读写权限。
- 如果程序出现异常，请检查输入数据的合法性，并根据提示信息进行修正。

---

6. 测试结果

测试数据

正确输入

- 添加任务：名称为“学习数据结构”，描述为“完成实验报告”，优先级为5，截止日期为2025-10-15。 **输出**：任务添加成功。
- 查询任务：名称为“学习数据结构”。 **输出**：任务信息，包括名称、描述、优先级和截止日期。

错误输入

- 添加任务：优先级为6（超出范围）。 **输出**：优先级不合法。
- 查询任务：名称为“不存在的任务”。 **输出**：任务未找到。

---

7. 附录

源代码

Task.h

```
#pragma once

#include <string>

// 定义任务的数据结构
struct Task {
    std::string name;         // 任务名称
    std::string description; // 任务描述
    int priority;             // 任务优先级（1-5，数字越大优先级越高）
    std::string dueDate;      // 任务截止日期（格式：YYYY-MM-DD）
};


// Notes:  为什么只有int priority前面没有std::string
// 因为int是基本数据类型，不需要包含头文件，而std::string是C++标准库中的类，需要包含<string>头文件。
```

TaskManager.h

```
#pragma once
```

```cpp
#include <string>
#include <vector>
#include "Task.h"

// 定义任务管理器的抽象基类
class TaskManager {
public:
    // 虚析构函数，确保通过基类指针删除派生类对象时能正确释放资源
    virtual ~TaskManager() noexcept = default;

    // 添加任务
    virtual void addTask(const Task& task) = 0;

    // 根据任务名称删除任务
    virtual bool deleteTask(const std::string& name) = 0;

    // 修改任务
    virtual bool updateTask(const std::string& name, const Task& newTask)
= 0;

    // 查询任务（根据名称或截止日期）
    virtual std::vector<Task> queryTasks(const std::string& queryString,
bool byName) = 0;

    // 获取所有任务并排序（按截止日期或优先级）
    virtual std::vector<Task> getAllTasksSorted(bool byDueDate) = 0;

    // 保存任务列表到文件
    virtual bool saveToFile(const std::string& filename) = 0;

    // 从文件读取任务列表
    virtual bool loadFromFile(const std::string& filename) = 0;
};
```

main.cpp

```cpp
#include <iostream>
#include <string>
#include <limits>   // 提供数值极限
#include <vector>
#include <chrono>   // 获取当前时间
#include <ctime>    // 日期计算
#include <sstream>  // 解析日期字符串
#include <cctype>   // 字符检测支持

#include "TaskManager.h"
#include "SequentialTaskManager.h"    // 顺序表实现
#include "LinkedTaskManager.h"        // 单向链表实现
#include "DoublyLinkedTaskManager.h" // 双向链表实现
```

```
// --- 原有的函数声明 ---
void printMenu();
void handleAddTask(TaskManager* manager);
void handleDeleteTask(TaskManager* manager);
void handleUpdateTask(TaskManager* manager);
void handleQueryTasks(TaskManager* manager);
void handleDisplayTasks(TaskManager* manager, bool byDueDate);
void handleSaveToFile(TaskManager* manager);
void handleLoadFromFile(TaskManager* manager);
void handleCheckUpcomingTasks(TaskManager* manager);
void printTasks(const std::vector<Task>& tasks);
std::vector<Task> filterUpcomingTasks(const std::vector<Task>& allTasks,
int daysThreshold);
bool isValidDate(const std::string& dateStr);
bool isValidPriority(int priority);

int main() {
    TaskManager* manager = nullptr;
    int choice;

    std::cout << "========================================\n";
    std::cout << "个人任务管理系统\n";
    std::cout << "========================================\n";

    std::cout << "请选择数据结构 (1: 顺序表, 2: 单向链表, 3: 双向链表): ";
    std::cin >> choice;

    if (choice == 1) {
        manager = new SequentialTaskManager();
        std::cout << "\n已选择 [顺序表] 实现。\n";
    } else if (choice == 2) {
        manager = new LinkedTaskManager();
        std::cout << "\n已选择 [单向链表] 实现。\n";
    } else if (choice == 3) {
        manager = new DoublyLinkedTaskManager();
        std::cout << "\n已选择 [双向链表] 实现。\n";
    }
    else {
        std::cout << "无效选择，程序退出。\n";
        return 1;
    }

    // 主循环
    while (true) {
        printMenu();
        std::cout << "请输入您的选择: ";
        std::cin >> choice;

        if (std::cin.fail()) {
            std::cout << "错误：请输入一个数字。\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
```

```
            continue;
        }

        switch (choice) {
            case 1: handleAddTask(manager); break;
            case 2: handleDeleteTask(manager); break;
            case 3: handleUpdateTask(manager); break;
            case 4: handleQueryTasks(manager); break;
            case 5: handleDisplayTasks(manager, true); break;  // true:  按
ddl排序
            case 6: handleDisplayTasks(manager, false); break; // false: 按
优先级排序
            case 7: handleSaveToFile(manager); break;
            case 8: handleLoadFromFile(manager); break;
            case 9: handleCheckUpcomingTasks(manager); break;
            case 0:
                std::cout << "感谢使用，再见！\n";
                delete manager;
                return 0;
            default:
                std::cout << "无效选择，请重新输入。\n";
                break;
        }
    }

    return 0;
}

// 打印主菜单
void printMenu() {
    std::cout << "\n---------- 主菜单 ----------\n";
    std::cout << "1. 添加新任务\n";
    std::cout << "2. 删除任务\n";
    std::cout << "3. 修改任务\n";
    std::cout << "4. 查询任务\n";
    std::cout << "5. 按截止日期显示所有任务\n";
    std::cout << "6. 按优先级显示所有任务\n";
    std::cout << "7. 保存任务到文件\n";
    std::cout << "8. 从文件加载任务\n";
    std::cout << "9. 检查即将到期的任务\n";
    std::cout << "0. 退出系统\n";
    std::cout << "--------------------------\n";
}


void cleanInputBuffer() {
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

void printTasks(const std::vector<Task>& tasks) {
    if (tasks.empty()) {
        std::cout << "任务列表为空或未找到匹配项。\n";
        return;
    }
```

```
    std::cout << "-------------------------------------------------\n";
    for (const Task& task : tasks) {
        std::cout << "任务名称: " << task.name << "\n"
                  << "  描述: " << task.description << "\n"
                  << "  优先级: " << task.priority << "\n"
                  << "  截止日期: " << task.dueDate << "\n";
        std::cout << "-------------------------------------------------
\n";
    }
}

void handleAddTask(TaskManager* manager) {
    Task newTask;
    cleanInputBuffer();

    while (true) {
        std::cout << "请输入任务名称: ";
        std::getline(std::cin, newTask.name);
        if (newTask.name.empty()) {
            std::cout << "任务名称不能为空。\n";
            continue;
        }
        if (!manager->queryTasks(newTask.name, true).empty()) {
            std::cout << "存在同名任务，请输入其他名称。\n";
            continue;
        }
        break;
    }

    std::cout << "请输入任务描述: ";
    std::getline(std::cin, newTask.description);

    int priorityInput;
    while (true) {
        std::cout << "请输入优先级(1-5)(数字越大表示越紧急): ";
        if (!(std::cin >> priorityInput)) {
            std::cout << "输入错误：优先级必须是数字。\n";
            std::cin.clear();
            cleanInputBuffer();
            continue;
        }
        if (!isValidPriority(priorityInput)) {
            std::cout << "输入错误：优先级范围为 1-5。\n";
            cleanInputBuffer();
            continue;
        }
        break;
    }

    std::string dueDateInput;
    while (true) {
        std::cout << "请输入截止时间(YYYY-MM-DD): ";
        std::cin >> dueDateInput;
        if (!isValidDate(dueDateInput)) {
```

```
            std::cout << "输入错误：截止日期必须符合 YYYY-MM-DD 格式且为有效日期。
\n";
            continue;
        }
        break;
    }

    newTask.priority = priorityInput;
    newTask.dueDate = dueDateInput;
    manager->addTask(newTask);
    std::cout << "添加成功！\n";
}

void handleDeleteTask(TaskManager* manager) {
    std::string name;
    cleanInputBuffer();
    std::cout << "请输入要删除的任务名称：";
    std::getline(std::cin, name);   // getline 可以读取包含空格的任务名 cin不行
    manager->deleteTask(name);
}

void handleUpdateTask(TaskManager* manager) {
    std::string name;
    cleanInputBuffer();
    std::cout << "请输入要修改的任务名称：";
    std::getline(std::cin, name);

    if (manager->queryTasks(name, true).empty()) {
        std::cout << "未找到该任务。\n";
        return;
    }

    Task updatedTask;
    updatedTask.name = name;
    std::cout << "请输入新的任务描述：";
    std::getline(std::cin, updatedTask.description);
    std::cout << "请输入新的优先级 (1-5)：";
    std::cin >> updatedTask.priority;
    std::cout << "请输入新的截止日期 (YYYY-MM-DD)：";
    std::cin >> updatedTask.dueDate;

    if (manager->updateTask(name, updatedTask)) {
        std::cout << "修改成功！\n";
    }
}

void handleQueryTasks(TaskManager* manager) {
    int choice;
    std::cout << "请选择查询方式 (1: 按名称, 2: 按截止日期)：";
    std::cin >> choice;
    cleanInputBuffer();

    std::string query;
    std::vector<Task> results;
```

```
    if (choice == 1) {
        std::cout << "请输入任务名称: ";
        std::getline(std::cin, query);
        results = manager->queryTasks(query, true);
    } else if (choice == 2) {
        std::cout << "请输入截止日期 (YYYY-MM-DD): ";
        std::getline(std::cin, query);
        results = manager->queryTasks(query, false);
    } else {
        std::cout << "无效选择。\n";
        return;
    }
    std::cout << "\n查询结果:\n";
    printTasks(results);
}

void handleDisplayTasks(TaskManager* manager, bool byDueDate) {
    std::vector<Task> tasks = manager->getAllTasksSorted(byDueDate);
    if (byDueDate) {
        std::cout << "\n---所有任务 (按截止日期排序)---\n";
    } else {
        std::cout << "\n---所有任务 (按优先级排序)---\n";
    }
    printTasks(tasks);
}

void handleSaveToFile(TaskManager* manager) {
    std::string filename;
    cleanInputBuffer();
    std::cout << "请输入要保存的文件名 (例如: tasks.txt): ";
    std::getline(std::cin, filename);
    if (manager->saveToFile(filename)) {
        std::cout << "保存成功! \n";
    } else {
        std::cout << "保存失败! \n";
    }
}

void handleLoadFromFile(TaskManager* manager) {
    std::string filename;
    cleanInputBuffer();
    std::cout << "请输入要导入的任务文件地址: ";
    std::getline(std::cin, filename);
    if (manager->loadFromFile(filename)) {
        std::cout << "导入成功! \n";
    } else {
        std::cout << "文件不存在，导入失败! \n";
    }
}

void handleCheckUpcomingTasks(TaskManager* manager) {
    int daysThreshold;
    std::cout << "请输入要检查的天数 (例如，输入 7 会查找未来一周内到期的任务): ";
```

```
    std::cin >> daysThreshold;

    // 检查用户输入是否为有效数字
    if (std::cin.fail() || daysThreshold < 0) {
        std::cout << "错误：请输入一个有效的非负整数。\n";
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
        return;
    }

    std::cout << "\n--- 检查未来 " << daysThreshold << " 天内到期的任务 ---
\n";

    // 步骤 1: 从 manager 获取所有任务
    std::vector<Task> allTasks = manager->getAllTasksSorted(false);

    // 步骤 2: 将所有任务和用户输入的天数交给独立的筛选函数处理
    std::vector<Task> upcomingTasks = filterUpcomingTasks(allTasks,
daysThreshold);

    printTasks(upcomingTasks);
}

bool isValidPriority(int priority) {
    return priority >= 1 && priority <= 5;
}

bool isValidDate(const std::string& dateStr) {
    if (dateStr.size() != 10 || dateStr[4] != '-' || dateStr[7] != '-') {
        return false;
    }

    for (std::size_t i = 0; i < dateStr.size(); ++i) {
        if (i == 4 || i == 7) {
            continue;
        }
        if (!std::isdigit(static_cast<unsigned char>(dateStr[i]))) {
            return false;
        }
    }

    int year = std::stoi(dateStr.substr(0, 4));
    int month = std::stoi(dateStr.substr(5, 2));
    int day = std::stoi(dateStr.substr(8, 2));

    if (month < 1 || month > 12) {
        return false;
    }

    const int daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31};
    int maxDay = daysInMonth[month - 1];
```

```
    bool isLeap = (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
    if (month == 2 && isLeap) {
        maxDay = 29;
    }

    return day >= 1 && day <= maxDay;
}

// 输入: "YYYY-MM-DD" 格式的日期字符串 返回: 该日期距离今天的天数。负数表示已过期。
int daysUntil(const std::string& dateStr) {
    std::tm dueDate = {};
    std::stringstream ss(dateStr);
    char delimiter;
    int year, month, day;

    ss >> year >> delimiter >> month >> delimiter >> day;
    dueDate.tm_year = year - 1900;
    dueDate.tm_mon = month - 1;
    dueDate.tm_mday = day;

    auto now = std::chrono::system_clock::now();
    std::time_t now_time = std::chrono::system_clock::to_time_t(now);
    std::tm* today = std::localtime(&now_time);

    std::tm today_date_only = *today;
    today_date_only.tm_hour = 0; today_date_only.tm_min = 0;
today_date_only.tm_sec = 0;

    std::time_t due_time = std::mktime(&dueDate);
    std::time_t today_time = std::mktime(&today_date_only);

    if (due_time == -1) return -9999; // 无效日期

    double seconds_diff = std::difftime(due_time, today_time);
    return static_cast<int>(seconds_diff / (60 * 60 * 24));
}

// 独立的提醒服务函数（与数据结构无关）
std::vector<Task> filterUpcomingTasks(const std::vector<Task>& allTasks,
int daysThreshold) {
    std::vector<Task> result;
    for (const auto& task : allTasks) {
        int remainingDays = daysUntil(task.dueDate);
        if (remainingDays >= 0 && remainingDays <= daysThreshold) {
            result.push_back(task);
        }
    }
    return result;
}
```

SequentialTaskManager.h

```
#pragma once

#include "TaskManager.h"
#include <vector>

// 派生类：使用顺序表 (std::vector) 实现任务管理器
class SequentialTaskManager : public TaskManager {
public:
    // 重写 (override) 基类的所有纯虚函数
    void addTask(const Task& task) override;
    bool deleteTask(const std::string& name) override;
    bool updateTask(const std::string& name, const Task& newTask)
override;
    std::vector<Task> queryTasks(const std::string& queryString, bool
byName) override;
    std::vector<Task> getAllTasksSorted(bool byDueDate) override;
    bool saveToFile(const std::string& filename) override;
    bool loadFromFile(const std::string& filename) override;

private:
    std::vector<Task> tasks; // 使用 std::vector 来存储任务
};
```

SequentialTaskManager.cpp

```
#include "SequentialTaskManager.h"
#include <fstream>
#include <algorithm>
#include <iostream>

void SequentialTaskManager::addTask(const Task& task) {
    tasks.push_back(task);
}

bool SequentialTaskManager::deleteTask(const std::string& name) {
    auto it = std::find_if(tasks.begin(), tasks.end(), [&](const Task&
task) {
        return task.name == name;
    });

    if (it != tasks.end()) {
        tasks.erase(it);
        return true;
    }

    std::cout << "不存在该任务，无法删除。\n";
    return false;
}
```

```
bool SequentialTaskManager::updateTask(const std::string& name, const
Task& newTask) {
    auto it = std::find_if(tasks.begin(), tasks.end(), [&](const Task&
task) {
        return task.name == name;
    });

    if (it != tasks.end()) {
        *it = newTask;
        return true;
    }

    return false;
}

std::vector<Task> SequentialTaskManager::queryTasks(const std::string&
queryString, bool byName) {
    std::vector<Task> result;
    for (const auto& task : tasks) {
        if (byName) {
            if (task.name == queryString) {
                result.push_back(task);
            }
        } else {
            if (task.dueDate == queryString) {
                result.push_back(task);
            }
        }
    }
    return result;
}

std::vector<Task> SequentialTaskManager::getAllTasksSorted(bool byDueDate)
{
    std::vector<Task> sortedTasks = tasks;

    if (byDueDate) {
        std::sort(sortedTasks.begin(), sortedTasks.end(), [](const Task&
a, const Task& b) {
            return a.dueDate < b.dueDate;
        });
    } else {
        std::sort(sortedTasks.begin(), sortedTasks.end(), [](const Task&
a, const Task& b) {
            return a.priority > b.priority;
        });
    }
    return sortedTasks;
}

bool SequentialTaskManager::saveToFile(const std::string& filename) {
    std::ofstream outFile(filename);
    if (!outFile.is_open()) {
        return false;
```

```
    }

    for (const auto& task : tasks) {
        outFile << task.name << "," << task.description << "," <<
task.priority << "," << task.dueDate << "\n";
    }

    outFile.close();
    return true;
}

bool SequentialTaskManager::loadFromFile(const std::string& filename) {
    std::ifstream inFile(filename);
    if (!inFile.is_open()) {
        return false;
    }

    tasks.clear();
    std::string line;
    while (std::getline(inFile, line)) {
        Task task;
        size_t pos = 0;
        std::string parts[4];

        for(int i=0; i<3; ++i) {
            pos = line.find(",");
            parts[i] = line.substr(0, pos);
            line.erase(0, pos + 1);
        }
        parts[3] = line;

        task.name = parts[0];
        task.description = parts[1];
        task.priority = std::stoi(parts[2]);
        task.dueDate = parts[3];

        tasks.push_back(task);
    }

    inFile.close();
    return true;
}
```

LinkedTaskManager.h

```
#pragma once

#include "TaskManager.h"

// 派生类：使用单向链表实现任务管理器
class LinkedTaskManager : public TaskManager {
```

```
public:
    // 构造函数，初始化头指针
    LinkedTaskManager();
    // 析构函数，释放链表所有节点的内存
    ~LinkedTaskManager();

    // 重写基类的所有纯虚函数
    void addTask(const Task& task) override;
    bool deleteTask(const std::string& name) override;
    bool updateTask(const std::string& name, const Task& newTask)
override;
    std::vector<Task> queryTasks(const std::string& queryString, bool
byName) override;
    std::vector<Task> getAllTasksSorted(bool byDueDate) override;
    bool saveToFile(const std::string& filename) override;
    bool loadFromFile(const std::string& filename) override;

private:
    struct Node {
        Task data;
        Node* next;
        Node(const Task& task) : data(task), next(nullptr) {}
    };

    Node* head; // 指向链表头部的指针

    // 清空链表函数
    void clear();
};
```

LinkedTaskManager.cpp

```
#include "LinkedTaskManager.h"
#include <fstream>
#include <algorithm>
#include <iostream>

// 构造函数
LinkedTaskManager::LinkedTaskManager() : head(nullptr) {}

// 析构函数
LinkedTaskManager::~LinkedTaskManager() {
    clear();
}

// 私有辅助函数：清空链表
void LinkedTaskManager::clear() {
    Node* current = head;
    while (current != nullptr) {
```

```
        Node* next = current->next;
        delete current;
        current = next;
    }
    head = nullptr;
}

// 1. 添加任务
void LinkedTaskManager::addTask(const Task& task) {
    Node* newNode = new Node(task);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* current = head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newNode;
    }
}

// 2. 删除任务
bool LinkedTaskManager::deleteTask(const std::string& name) {
    if (head == nullptr) return false;

    // 特殊情况：要删除的是头节点
    if (head->data.name == name) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return true;
    }

    Node* current = head;
    while (current->next != nullptr && current->next->data.name != name) {
        current = current->next;
    }

    if (current->next != nullptr) { // 找到了
        Node* temp = current->next;
        current->next = temp->next;
        delete temp;
        return true;
    }

    std::cout << "提示: 未找到名为 '" << name << "' 的任务，无法删除。\n";
    return false; // 未找到
}

// 3. 修改任务
bool LinkedTaskManager::updateTask(const std::string& name, const Task&
newTask) {
    Node* current = head;
    while (current != nullptr) {
```

```
        if (current->data.name == name) {
            current->data = newTask;
            return true;
        }
        current = current->next;
    }
    return false;
}

// 4. 查询任务
std::vector<Task> LinkedTaskManager::queryTasks(const std::string&
queryString, bool byName) {
    std::vector<Task> result;
    Node* current = head;
    while (current != nullptr) {
        bool match = byName ? (current->data.name == queryString) :
(current->data.dueDate == queryString);
        if (match) {
            result.push_back(current->data);
        }
        current = current->next;
    }
    return result;
}

// 5. 查看任务列表（排序）
std::vector<Task> LinkedTaskManager::getAllTasksSorted(bool byDueDate) {
    std::vector<Task> allTasks;
    Node* current = head;
    // 步骤1: 将链表中的所有任务复制到 vector 中
    while (current != nullptr) {
        allTasks.push_back(current->data);
        current = current->next;
    }

    // 步骤2: 对 vector 进行排序（与 SequentialTaskManager 完全相同的逻辑）
    if (byDueDate) {
        std::sort(allTasks.begin(), allTasks.end(), [](const Task& a,
const Task& b) {
            return a.dueDate < b.dueDate;
        });
    } else {
        std::sort(allTasks.begin(), allTasks.end(), [](const Task& a,
const Task& b) {
            return a.priority > b.priority;
        });
    }
    return allTasks;
}

// 6. 保存到文件
bool LinkedTaskManager::saveToFile(const std::string& filename) {
    std::ofstream outFile(filename);
    if (!outFile.is_open()) return false;
```

```
    Node* current = head;
    while (current != nullptr) {
        outFile << current->data.name << "," << current->data.description
<< ","
                << current->data.priority << "," << current->data.dueDate
<< "\n";
        current = current->next;
    }

    outFile.close();
    return true;
}

// 7. 从文件加载
bool LinkedTaskManager::loadFromFile(const std::string& filename) {
    std::ifstream inFile(filename);
    if (!inFile.is_open()) return false;

    clear(); // 加载前先清空现有链表

    std::string line;
    while (std::getline(inFile, line)) {
        Task task;
        size_t pos = 0;

        // 简单的CSV解析
        pos = line.find(",");
        task.name = line.substr(0, pos);
        line.erase(0, pos + 1);

        pos = line.find(",");
        task.description = line.substr(0, pos);
        line.erase(0, pos + 1);

        pos = line.find(",");
        task.priority = std::stoi(line.substr(0, pos));
        line.erase(0, pos + 1);

        task.dueDate = line;

        addTask(task); // 直接调用 addTask 方法来添加到链表
    }

    inFile.close();
    return true;
}
```

DoublyLinkedTaskManager.h

```
#pragma once

#include "TaskManager.h"

// 派生类：使用双向链表实现任务管理器
class DoublyLinkedTaskManager : public TaskManager {
public:
    // 构造函数
    DoublyLinkedTaskManager();
    // 析构函数
    ~DoublyLinkedTaskManager();

    // 重写基类的所有纯虚函数
    void addTask(const Task& task) override;
    bool deleteTask(const std::string& name) override;
    bool updateTask(const std::string& name, const Task& newTask)
override;
    std::vector<Task> queryTasks(const std::string& queryString, bool
byName) override;
    std::vector<Task> getAllTasksSorted(bool byDueDate) override;
    bool saveToFile(const std::string& filename) override;
    bool loadFromFile(const std::string& filename) override;

private:
    // 双向链表节点结构
    struct Node {
        Task data;
        Node* next;
        Node* prev; // 新增：指向前一个节点的指针
        Node(const Task& task) : data(task), next(nullptr), prev(nullptr)
{}
    };

    Node* head; // 指向链表头部
    Node* tail; // 指向链表尾部

    // 清空链表函数
    void clear();
};
```

DoublyLinkedTaskManager.cpp

```
#include "DoublyLinkedTaskManager.h"
#include <fstream>
#include <algorithm>
#include <iostream>

// 构造函数
DoublyLinkedTaskManager::DoublyLinkedTaskManager() : head(nullptr),
```

```
tail(nullptr) {}

// 析构函数
DoublyLinkedTaskManager::~DoublyLinkedTaskManager() {
    clear();
}

// 私有辅助函数：清空链表
void DoublyLinkedTaskManager::clear() {
    Node* current = head;
    while (current != nullptr) {
        Node* nextNode = current->next;
        delete current;
        current = nextNode;
    }
    head = nullptr;
    tail = nullptr;
}

// 1. 添加任务（得益于tail指针，效率更高）
void DoublyLinkedTaskManager::addTask(const Task& task) {
    Node* newNode = new Node(task);
    if (head == nullptr) { // 链表为空
        head = newNode;
        tail = newNode;
    } else { // 链表不为空
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode; // 更新尾指针
    }
}

// 2. 删除任务（指针操作更复杂）
bool DoublyLinkedTaskManager::deleteTask(const std::string& name) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data.name == name) {
            // 找到了要删除的节点
            if (current->prev != nullptr) {
                current->prev->next = current->next;
            } else { // 是头节点
                head = current->next;
            }

            if (current->next != nullptr) {
                current->next->prev = current->prev;
            } else { // 是尾节点
                tail = current->prev;
            }

            delete current;
            std::cout << "任务 '" << name << "' 删除成功！\n";
            return true;
        }
```

```
        current = current->next;
    }

    std::cout << "不存在该任务，无法删除。\n";
    return false;
}

// 3. 修改任务
bool DoublyLinkedTaskManager::updateTask(const std::string& name, const
Task& newTask) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data.name == name) {
            current->data = newTask;
            return true;
        }
        current = current->next;
    }
    return false;
}

// 4. 查询任务
std::vector<Task> DoublyLinkedTaskManager::queryTasks(const std::string&
queryString, bool byName) {
    std::vector<Task> result;
    Node* current = head;
    while (current != nullptr) {
        bool match = byName ? (current->data.name == queryString) :
(current->data.dueDate == queryString);
        if (match) {
            result.push_back(current->data);
        }
        current = current->next;
    }
    return result;
}

// 5. 查看任务列表（排序） - 逻辑与单向链表实现相同
std::vector<Task> DoublyLinkedTaskManager::getAllTasksSorted(bool
byDueDate) {
    std::vector<Task> allTasks;
    Node* current = head;
    while (current != nullptr) {
        allTasks.push_back(current->data);
        current = current->next;
    }

    if (byDueDate) {
        std::sort(allTasks.begin(), allTasks.end(), [](const Task& a,
const Task& b) {
            return a.dueDate < b.dueDate;
        });
    } else {
        std::sort(allTasks.begin(), allTasks.end(), [](const Task& a,
```

```
const Task& b) {
            return a.priority > b.priority;
        });
    }
    return allTasks;
}

// 6. 保存到文件 - 逻辑与单向链表实现相同
bool DoublyLinkedTaskManager::saveToFile(const std::string& filename) {
    std::ofstream outFile(filename);
    if (!outFile.is_open()) return false;

    Node* current = head;
    while (current != nullptr) {
        outFile << current->data.name << "," << current->data.description
<< ","
                << current->data.priority << "," << current->data.dueDate
<< "\n";
        current = current->next;
    }
    outFile.close();
    return true;
}

// 7. 从文件加载
bool DoublyLinkedTaskManager::loadFromFile(const std::string& filename) {
    std::ifstream inFile(filename);
    if (!inFile.is_open()) return false;

    clear(); // 加载前先清空

    std::string line;
    while (std::getline(inFile, line)) {
        Task task;
        size_t pos = 0;

        pos = line.find(",");
        task.name = line.substr(0, pos);
        line.erase(0, pos + 1);

        pos = line.find(",");
        task.description = line.substr(0, pos);
        line.erase(0, pos + 1);

        pos = line.find(",");
        task.priority = std::stoi(line.substr(0, pos));
        line.erase(0, pos + 1);

        task.dueDate = line;

        addTask(task); // 调用addTask来添加
    }

    inFile.close();
```

```
        return true;
    }
```