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

程序目标

本实验实现一个任务管理小程序。用户可以选择不同的数据结构完成任务的增删改查、排序、存盘与读盘等常用操作，并比较顺序表、单链表和双链表的差异。通过统一的功能接口，可以更直观地理解三种线性结构在实现细节和效率上的取舍。

输入范围

- **名称**：不超过 50 个字符的字符串。
- **描述**：不超过 200 个字符的字符串。
- **优先级**：整数 1~5，值越大表示越紧急。
- **截止日期**：采用 `YYYY-MM-DD` 格式的日期字符串。

输出内容

- **任务列表**：展示名称、描述、优先级和截止日期。
- **提示信息**：如操作成功与否的反馈。

功能划分

- 添加、删除、修改、查询任务。
- 按截止日期或优先级对任务进行排序。
- 将任务列表保存到磁盘或再次加载。

测试

- 例：添加“数据结构作业”，优先级 5，截止日期 2025-10-15，再按名称查询该任务。
- 例：输入优先级 6；或者查询从未录入的任务名称。

2. 概要设计

抽象数据结构

- `Task` 表示单个任务，封装名称、描述、优先级和截止日期。
- `TaskManager` 为抽象基类，给出统一的任务管理接口。

总体流程

1. 程序启动后让用户在顺序表、单链表、双链表三种实现里任选其一。
2. 显示菜单，用户依次执行增删改查、排序、存储等操作。
3. 操作完成后打印结果或提示信息。

模块关系

```
main.cpp
 ├─ TaskManager.h
 ├─ Task.h
 ├─ SequentialTaskManager.{h,cpp}
 ├─ LinkedTaskManager.{h,cpp}
 └─ DoublyLinkedTaskManager.{h,cpp}
```

3. 详细设计

核心数据类型

```
struct Task {
    std::string name;
    std::string description;
    int priority;
    std::string dueDate;
};
```

```
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;
};
```

关键功能逻辑

- **添加**：将新任务附加到当前结构末尾或尾节点。
- **删除 / 修改**：线性扫描查找目标，找到后执行删除或覆盖。
- **查询**：按名称或截止日期遍历并收集匹配项。
- **排序**：顺序表用冒泡写法反复交换；链表实现则把数据挪到临时数组里，再用同样循环完成插入或选择排序。
- **持久化**：任务数据使用逗号分隔的文本格式保存。

---

4. 调试记录

1. **优先级越界**
   原因：缺乏输入范围校验。
   处理：在录入阶段增加范围判断并给出提示。

2. **重复任务名称**
   原因：添加时未检查重名。
   处理：插入前先遍历任务表，若已有同名任务则拒绝添加。

3. **文件存取失败**
   原因：路径错误或无权限。
   处理：文件打开失败时输出提示，并保持原数据不变。

4. **排序循环冗长**
   原因：为了图省事，直接写了最基础的双层循环，没有做任何提前退出或复用。
   处理：保留这种朴素写法，虽然多跑几轮，但逻辑一目了然，方便后续调试。

---

5. 使用说明

1. **编译**

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

2. **运行**

```
./TaskManager
```

按提示选择数据结构，再根据菜单输入指令。

3. **常见操作**
   - 添加：输入任务四要素。
   - 删除：输入任务名称。
   - 修改：输入名称后提供新内容。
   - 查询：按名称或日期检索。
   - 排序：可按日期升序，或按优先级降序。
   - 保存 / 加载：输入文件名即可。
4. **提示**
   - 日期需要符合 `YYYY-MM-DD`。
   - 保存文件前确认路径可写。

---

6. 测试情况

序号	场景描述	预期结果
1	添加任务“学习数据结构”，优先级 5，截止 2025-10-15	成功，列表中可见该任务
2	查询名称为“学习数据结构”的任务	打印完整任务信息
3	添加优先级 6 的任务	提示优先级非法，操作失败
4	查询不存在的任务名称	提示未找到

---

7. 附录

Task.h

```
#pragma once

#include <string>

struct Task {
    std::string name;
    std::string description;
    int priority;
    std::string dueDate;
};
```

TaskManager.h

```
#pragma once

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

class TaskManager {
public:
    virtual ~TaskManager() noexcept = default;
    virtual void addTask(const Task& task) = 0;
    virtual bool deleteTask(const std::string& name) = 0;
    virtual bool updateTask(const std::string& name, const Task& newTask)
= 0;
    virtual std::vector<Task> queryTasks(const std::string& queryString,
bool byName) = 0;
    virtual std::vector<Task> getAllTasksSorted(bool byDueDate) = 0;
    virtual bool saveToFile(const std::string& filename) = 0;
    virtual bool loadFromFile(const std::string& filename) = 0;
};
```

main.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


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.cpp

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

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

bool SequentialTaskManager::deleteTask(const std::string& name) {
    int length = static_cast<int>(tasks.size());
```

```
    for (int i = 0; i < length; ++i) {
        if (tasks[static_cast<std::size_t>(i)].name == name) {
            tasks.erase(tasks.begin() + i);
            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) {
    for (auto& task : tasks) {
        if (task.name == name) {
            task = 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 (sortedTasks.empty()) {
        return sortedTasks;
    }

    if (byDueDate) {
        int limit = static_cast<int>(sortedTasks.size());
        for (int i = 0; i < limit; ++i) {
            for (int j = 0; j < limit - 1; ++j) {
                std::size_t idx = static_cast<std::size_t>(j);
                std::size_t nextIdx = static_cast<std::size_t>(j + 1);
```

```
                if (sortedTasks[idx].dueDate >
sortedTasks[nextIdx].dueDate) {
                    Task temp = sortedTasks[idx];
                    sortedTasks[idx] = sortedTasks[nextIdx];
                    sortedTasks[nextIdx] = temp;
                }
            }
        }
    } else {
        int limit = static_cast<int>(sortedTasks.size());
        for (int i = 0; i < limit; ++i) {
            for (int j = 0; j < limit - 1; ++j) {
                std::size_t idx = static_cast<std::size_t>(j);
                std::size_t nextIdx = static_cast<std::size_t>(j + 1);
                if (sortedTasks[idx].priority <
sortedTasks[nextIdx].priority) {
                    Task temp = sortedTasks[idx];
                    sortedTasks[idx] = sortedTasks[nextIdx];
                    sortedTasks[nextIdx] = temp;
                }
            }
        }
    }

    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

```
#include "LinkedTaskManager.h"
#include <fstream>
#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;
    }

    if (byDueDate) {
        int len = static_cast<int>(allTasks.size());
        for (int i = 1; i < len; ++i) {
            Task currentTask = allTasks[static_cast<std::size_t>(i)];
            int back = i - 1;
            while (back >= 0) {
                std::size_t pos = static_cast<std::size_t>(back);
                if (currentTask.dueDate < allTasks[pos].dueDate) {
                    allTasks[pos + 1] = allTasks[pos];
                    --back;
                } else {
                    break;
                }
            }
            allTasks[static_cast<std::size_t>(back + 1)] = currentTask;
        }
    } else {
        int len = static_cast<int>(allTasks.size());
        for (int i = 1; i < len; ++i) {
            Task currentTask = allTasks[static_cast<std::size_t>(i)];
            int back = i - 1;
            while (back >= 0) {
                std::size_t pos = static_cast<std::size_t>(back);
                if (currentTask.priority > allTasks[pos].priority) {
                    allTasks[pos + 1] = allTasks[pos];
                    --back;
                } else {
                    break;
                }
            }
            allTasks[static_cast<std::size_t>(back + 1)] = currentTask;
```

```
        }
    }

    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

```
#include "DoublyLinkedTaskManager.h"
#include <fstream>
#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） - 采用自写选择排序
```

```cpp
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) {
        int total = static_cast<int>(allTasks.size());
        for (int i = 0; i < total; ++i) {
            int minIndex = i;
            for (int j = i + 1; j < total; ++j) {
                std::size_t check = static_cast<std::size_t>(j);
                std::size_t best = static_cast<std::size_t>(minIndex);
                if (allTasks[check].dueDate < allTasks[best].dueDate) {
                    minIndex = j;
                }
            }
            if (minIndex != i) {
                std::size_t first = static_cast<std::size_t>(i);
                std::size_t second = static_cast<std::size_t>(minIndex);
                Task temp = allTasks[first];
                allTasks[first] = allTasks[second];
                allTasks[second] = temp;
            }
        }
    } else {
        int total = static_cast<int>(allTasks.size());
        for (int i = 0; i < total; ++i) {
            int maxIndex = i;
            for (int j = i + 1; j < total; ++j) {
                std::size_t check = static_cast<std::size_t>(j);
                std::size_t best = static_cast<std::size_t>(maxIndex);
                if (allTasks[check].priority > allTasks[best].priority) {
                    maxIndex = j;
                }
            }
            if (maxIndex != i) {
                std::size_t first = static_cast<std::size_t>(i);
                std::size_t second = static_cast<std::size_t>(maxIndex);
                Task temp = allTasks[first];
                allTasks[first] = allTasks[second];
                allTasks[second] = temp;
            }
        }
    }

    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;
}
```