

XI`AN TECHNOLOGICAL UNIVERSITY

课程设计报告

课程名称：

专 业：

班 级：

姓 名：

学 号：

指导教师： 刘敏侠

成 绩：

2025 年 6 月 20 日

1. 问题描述

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | C语言项目实践 | | | | | 指导教师 | | 刘敏侠 | |
| 课设题目1 | 学生信息管理系统 | | | | | | | |  |
| 学号 |  | 姓名 |  | 地点 |  | | 完成时间 | | 2025/6/16  2025/6/20 |
| 设计任务 | 建立一个学生信息管理系统，利用单链表实现，具有如下的五大功能模块，具体要求如下：  qq_pic_merged_1687695704408  学生信息管理系统 | | | | | | | | |
| 设计要求 | 1. 输入记录模块   逐个输入学生基本信息，学生信息包含：学号、姓名、性别、年龄、班级等。   1. 查询记录模块   按照学生的学号或者姓名进行查询，把满足条件的学生信息显示出来。   1. 更新记录模块   修改某一个学生的信息，插入一个学生信息；删除一个学生的信息。   1. 统计记录模块   能够按班级、性别统计学生数量，并输出统计结果；。   1. 输出记录模块   显示所有学生的信息。   1. 能够将所有学生信息保存到文件中，并能从文件中读取学生信息；   以上为最基本要求，可根据自己的理解在此基础上对各个模块增加和完善相应的功能，使得整个系统能更加接近实际的应用。 | | | | | | | | |
| 成果要求 | 1. 设计出完整流程图。  2．编写出全部模块对应的函数。  3．在集成开发工具中调试通过。  4．参加答辩。  5．写出完整、规范的课程设计报告，在指定的时间内提交指导教师。 | | | | | | | | |
| 设计报告  要求 | 1．封面：（格式附后）  2．课程设计任务书  3．课程设计报告：  ①题目用黑体三号，段后距18磅（或1行），居中对齐；  ②标题用黑体四号，段前、段后距6磅（或0.3行）；  ③正文用小四号宋体，行距为1.25倍行距；  ④标题按“一”、“㈠”、“1”、“⑴”顺序编号。  ⑤报告和软件打包以学号姓名命名，邮件主题也为学号姓名。  ⑴ 系统总体方案  ①题目用黑体三号，段后距18磅（或1行），居中对齐；  ②标题用黑体四号，段前、段后距6磅（或0.3行）；  ③正文用小四号宋体，行距为1.25倍行距；  ④标题按“一”、“㈠”、“1”、“⑴”顺序编号。  ⑤报告和软件打包以学号姓名命名，邮件主题也为学号姓名。  ①题目用黑体三号，段后距18磅（或1行），居中对齐；  ②标题用黑体四号，段前、段后距6磅（或0.3行）；  ③正文用小四号宋体，行距为1.25倍行距；  ④标题按“一”、“㈠”、“1”、“⑴”顺序编号。  ⑤报告和软件打包以学号姓名命名，邮件主题也为学号姓名。  ⑵ 设计思路和主要步骤  ⑶ 各功能模块和流程图  ⑷ 设计代码  ⑸ 心得体会 | | | | | | | | |

1. 算法设计

本学生档案管理系统采用模块化的设计思路，其核心是围绕一个内存中的单向链表进行数据操作，并设计了一套实时的数据持久化机制，确保数据的安全性和一致性。

#### **1. 数据结构设计**

系统的基础数据模型在头文件 record\_manager.h 中定义，主要包含两个结构体：**Info\_Student**：该结构体作为基本的数据单元，用于封装一名学生的完整信息，包括学号 stu\_id、姓名 stu\_name、性别 stu\_gender、年龄 stu\_age 和班级 stu\_class。**Link\_Node**：这是构成链表的节点结构。每个节点包含一个 Info\_Student类型的 record 数据域和一个指向下一个节点的指针 p\_next。系统采用带哨兵（Sentinel）头结点的单链表，即链表头部有一个不存储实际数据的哑节点（dummy node），其 p\_next 指针指向第一个有效的学生记录节点。这一设计统一了链表操作，简化了在链表头部插入或删除节点的算法逻辑。

#### **2. 程序流程与实时持久化策略**

程序的整体执行流程由 main 函数驱动，其显著特点是采用了实时保存的持久化策略。

**启动与加载**：程序启动时，立即调用 load\_records\_from\_disk 函数，从磁盘文件 student\_data.txt 中读取全部记录，并在内存中构建起一个完整的链表。

**主交互循环**：程序进入一个 do-while 循环，循环体内展示主菜单并等待用户输入指令。

**数据修改与实时保存**：

当用户选择新增学生档案（指令1）时，程序在调用 create\_new\_record 成功添加记录到内存链表后，会 **立即** 调用 persist\_records\_to\_disk 函数，将最新的链表数据完整地写回磁盘文件。当用户选择更改学生档案（指令3）并进入 alter\_record\_menu 子菜单后，无论是执行修改还是删除操作，在内存操作完成后，程序同样会 **立即** 调用 persist\_records\_to\_disk 函数进行存盘。这种实时保存策略确保了任何对数据的修改都会即时持久化，避免了因程序异常退出而导致的数据丢失。

**退出**：当用户选择退出（指令0）时，程序直接退出主循环，无需再次保存。最后调用 release\_all\_nodes 释放内存。

#### **3. 核心功能模块算法**

**创建记录 (create\_new\_record)**动态分配一个新的 Link\_Node 节点。

在获取用户输入的学号后，立即遍历链表进行唯一性检查。若学号已存在，则释放节点并返回错误信息。若学号唯一，则接收用户输入的其他信息，并将新节点通过尾部插入法追加到链表末端。

**检索记录 (find\_record\_menu)**

**按学号查找 (find\_by\_student\_id)**: 采用线性扫描算法。从链表的第一个有效节点开始遍历，找到匹配的学号后，打印信息并立即返回。

**按姓名查找 (find\_by\_student\_name)**: 同样是线性扫描，但为了找出所有同名学生，该算法会遍历整个链表，并统计和显示所有匹配的记录。

**修改与移除记录 (alter\_record\_menu)**

**修改 (update\_record)**: 算法首先根据学号查找到目标节点，若找到，则允许用户输入全新的数据直接覆盖节点内的 record 数据域。**移除 (remove\_record)**: 采用“前驱指针-当前指针”双指针法。prev 指针和 curr 指针协同遍历，当 curr 指向目标节点时，通过 prev->p\_next = curr->p\_next 将 curr 节点从链表中逻辑上断开，然后调用 free 释放其内存。

**数据汇总 (summarize\_data\_menu)**

无论是按班级还是按性别统计，其核心算法都是对链表的一次完整遍历。在遍历过程中，通过 strcmp 函数对每个节点的相应字段进行判断，并累加计数器。

**文件I/O算法**

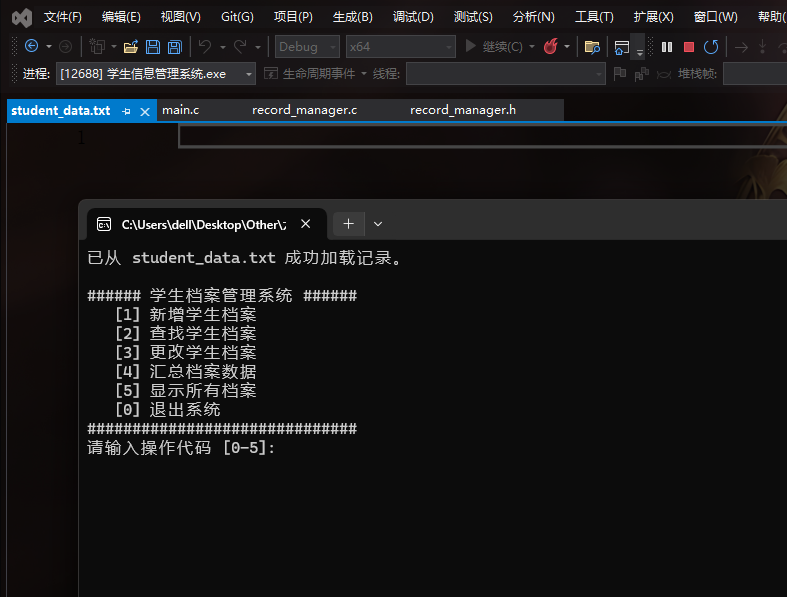
**保存 (persist\_records\_to\_disk)**: 这是一个 **数据序列化** 过程。函数以写入模式 ("w") 打开文件，此模式会清空文件内容。然后遍历链表，将每个节点的学生信息通过 fprintf 格式化为一行文本写入文件。

**加载 (load\_records\_from\_disk)**: 这是一个 **数据反序列化** 过程函数在读取文件时采用了一个高效的尾指针追加算法。它在初始化空链表的同时，维持一个 tail 指针始终指向链表的最后一个节点。当从文件成功读取一条记录并创建新节点后，直接通过 tail->p\_next = new\_node 进行追加，然后更新 tail 指针（tail = new\_node）。这避免了每次插入新节点时都从头遍历链表，将加载过程的时间复杂度优化为 O(N)，其中 N 是文件中的记录数。

**资源管理 (release\_all\_nodes)**

为防止内存泄漏，该函数负责释放程序运行期间动态分配的所有 Link\_Node 节点。它通过一个循环，从头结点开始，逐一释放每个节点占用的内存。

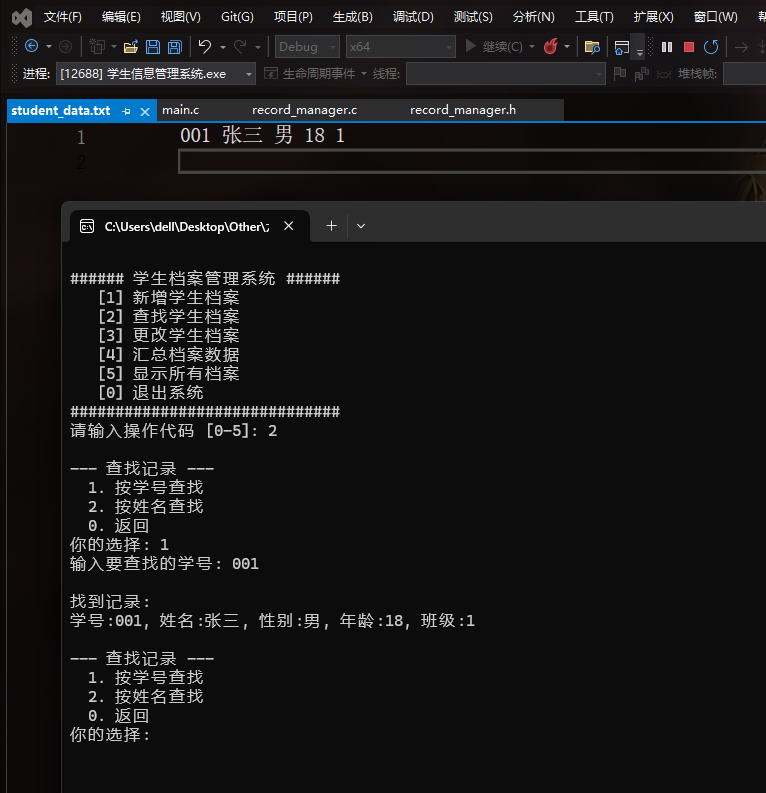
1. 算法实现
2. 程序主页面，启动与加载：程序启动后，首先显示从 student\_data.txt 文件加载数据的提示，然后展示主菜单，等待用户操作 。



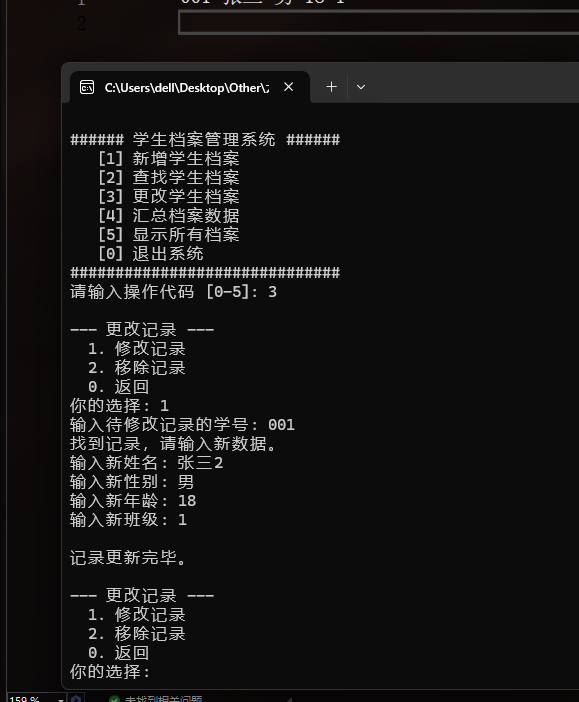
1. 录入学生信息，信息录入：下图展示了录入一个学号为"001"的学生信息的完整过程。添加成功后，系统提示数据已保存到文件 。



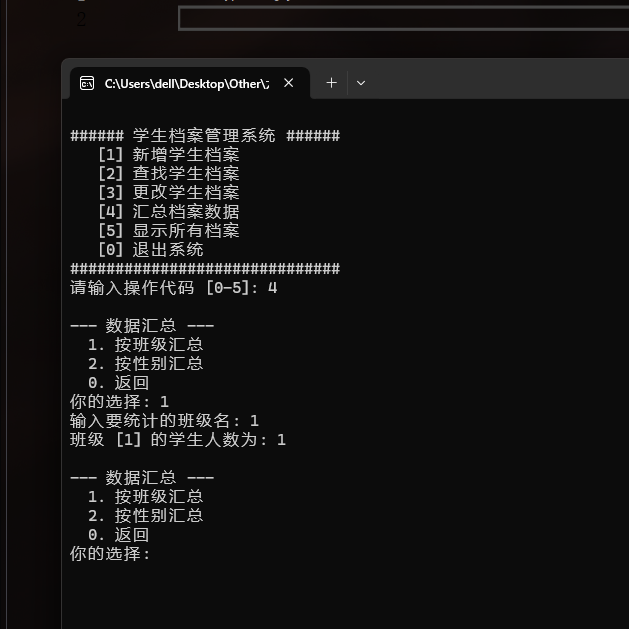
1. 查询学生信息，信息查询：系统提供了按学号和姓名两种查询方式。下图演示了分别使用这两种方式查询名为“张三”的学生，并准确返回了结果 。



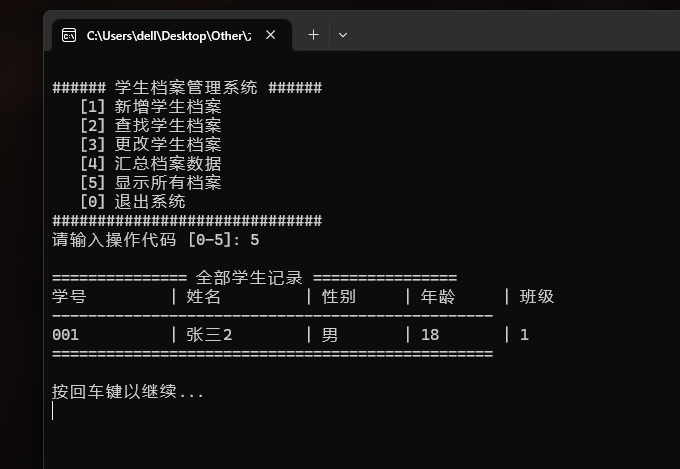
1. 更新学生信息，信息更新：用户可以修改或删除学生信息。下图展示了将学号"001"的学生的姓名从“张三”修改为“张三2”的操作。修改成功后，student\_data.txt 文件中的内容也同步更新



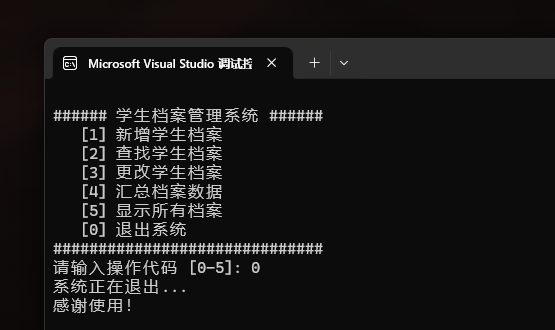
1. 统计学生信息，信息统计与显示：系统能够按班级和性别进行统计，也可以完整地列出所有学生的信息，方便用户全局浏览 。



1. 显示所有信息



1. 退出系统 Exit



四．心得体会

完成学生信息管理系统的课程设计，对我而言，其意义远超出于一次C语言的编程练习。我将它视为一次微缩版的软件工程实践，从需求分析、系统设计，到编码实现、测试交付，完整地体验了一个软件产品从无到有的全过程。在这个过程中，我最大的收获并非掌握了某个具体的C语言技巧，而是对软件的“架构”、“模块化”、“健壮性”和“用户体验”等工程化思想有了深刻的理解与认知。

**一、 模块化设计：告别“一锅炖”的优雅**

在项目启动之初，面对任务书上罗列的增、删、改、查、统计、保存、读取等一系列功能，我的第一反应是把所有代码都塞进main.c文件中。但很快，理智告诉我这将会是一场灾难。一个数千行的main.c文件，其逻辑必然错综复杂，变量名可能冲突，函数调用关系混乱，极难阅读、调试和维护。

因此，我坚决地采纳了模块化设计的思想，这也是本次设计给我上的最重要的一课。我将整个系统划分为三个清晰的模块（文件）：

**student.h (接口层)**：这是模块的“公共契约”。我在这里定义了核心的数据结构Student和Node，并声明了所有对外开放的函数原型，如addStudent、queryMenu等。任何需要使用这些功能的模块，只需要#include "student.h"即可。我还学习并使用了#ifndef...#define...#endif的头文件保护宏，防止了头文件被重复包含可能导致的编译错误。这让我理解了API设计的初步概念。

**student.c (实现层)**：这里是所有业务逻辑的具体实现。它包含了对链表操作、文件读写等所有核心功能的函数体。这一层只关心“如何做”，而将“做什么”的接口暴露在头文件中。这种接口与实现的分离，是软件工程的基石。它意味着，如果未来我希望将链表更换为二叉树以提高效率，我只需要修改student.c中的实现，而无需改动main.c中的调用代码，大大提高了系统的可维护性和可扩展性。

**main.c (表示/调用层)**：这个文件变得非常“轻”。它的职责很纯粹：展示用户菜单、接收用户输入、根据输入调用student.c中提供的服务函数，并处理主循环逻辑。它不关心数据是如何存储的，也不关心学生信息是如何被修改的，它只负责“调度”和“交互”。

通过这样的划分，我的项目结构一目了然。当我需要调试查询功能时，我能立刻定位到student.c中的searchStudentByName函数，而不会被主菜单的打印代码所干扰。这种“高内聚、低耦合”的设计思想，让我的开发过程井然有序，也让我深刻体会到良好架构带来的巨大便利。

**二、 健壮性（Robustness）：一个程序的“品格”**

一个能运行的程序和一个好的程序之间，隔着一条名为“健壮性”的鸿沟。在开发过程中，我花费了大量精力去思考和处理各种“意料之外”的情况，力求让我的程序具备良好的“品格”。

首先是用户输入的处理。C语言的scanf函数是一个著名的“陷阱”，如果用户输入的类型与格式化字符串不匹配，或者在输入数字后留下了多余的换行符，都可能导致后续的输入操作混乱甚至程序崩溃。为了解决这个问题，我编写了一个clearInputBuffer辅助函数，在每次调用scanf之后，都用它来清空输入缓冲区中多余的字符。此外，对于菜单选择，我也在switch语句中加入了default分支，用于处理用户输入0-5之外的无效选项，并给出友好提示。

其次是边界条件的处理。例如，当链表为空时，任何查询、修改、删除、统计操作都应该能够优雅地处理，而不是引发空指针异常。我在每个需要遍历链表的函数开头，都加入了if (head->next == NULL)的判断。同样，在加载文件时，如果数据文件尚不存在（首次运行），程序也应该能够正常启动，而不是报错退出。fopen返回NULL的情况被我视为一种正常的初始状态。这些对边界的细心处理，构成了程序健壮性的核心。

**三、 软件即服务：从“能用”到“好用”**

除了功能的实现和程序的稳定，我还思考了如何提升用户体验，即从“能用”到“好用”的转变。我精心设计了控制台的菜单界面，使用=和|等字符绘制出清晰的边框，让界面不那么单调。在每个操作完成后，我都会给出明确的成功或失败提示。在执行完一项操作后，我使用了printf("\n按 Enter 键继续..."); getchar();的组合，让程序暂停，等待用户看完结果后再清屏返回主菜单，而不是让信息一闪而过。这些微小的细节，共同营造了一个更加友好、更具体贴心的人机交互流程。

总结而言，这次课程设计让我跳出了纯粹“C语法”的禁锢，开始以一名“软件工程师”的视角来审视我的作品。我学会了在动手编码前先进行思考和设计，懂得了模块化、健壮性和用户体验对于一个软件项目的重要性。这些工程化的思想和方法，是我在本次实践中获得的最宝贵的财富，它们将对我未来的学习和职业生涯产生深远的影响。

五．参考文献

[1]张学孟,梅诗冬,吕家威.基于在线信息化平台的学生信息管理系统运用研究[J].信息系统工程,2025,(06):8-11.

[1]王英.C语言中循环转递归函数策略研究[J].科学技术创新,2025,(15):71-74.

[1]单宝华,佟智慧.C语言在线编程代码缺陷检测方法研究[J].网络安全技术与应用,2025,(05):56-59.

[1]夏黎晨.编程语言中的深拷贝和浅拷贝对算法设计的影响研究[J].软件,2025,46(04):151-153.

[1]葛方振,洪留荣.C语言程序设计基础实验教程[M].中国铁道出版社:202212.108.

源代码

1. #pragma once
2. #ifndef RECORD\_MANAGER\_H
3. #define RECORD\_MANAGER\_H
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. *// 学生信息*
8. typedef struct Info\_Student {
9. char stu\_id[20];
10. char stu\_name[50];
11. char stu\_gender[10];
12. int stu\_age;
13. char stu\_class[50];
14. } Info\_Student;
15. *// 链表节点*
16. typedef struct Link\_Node {
17. Info\_Student record;
18. struct Link\_Node\* p\_next;
19. } Link\_Node;
20. *// --- 接口声明 ---*
21. *// 文件操作*
22. Link\_Node\* load\_records\_from\_disk(const char\* filepath);
23. void persist\_records\_to\_disk(Link\_Node\* list\_head, const char\* filepath);
24. *// 核心功能*
25. void create\_new\_record(Link\_Node\* list\_head);
26. void find\_record\_menu(Link\_Node\* list\_head);
27. void alter\_record\_menu(Link\_Node\* list\_head);
28. void summarize\_data\_menu(Link\_Node\* list\_head);
29. void print\_all\_records(Link\_Node\* list\_head);
30. *// 资源管理*
31. void release\_all\_nodes(Link\_Node\* list\_head);
32. #endif *// RECORD\_MANAGER\_H*
33. #define \_CRT\_SECURE\_NO\_WARNINGS
34. #include "record\_manager.h"
35. static void flush\_stdin() {
36. int ch;
37. while ((ch = getchar()) != '\n' && ch != EOF);
38. }
39. *// 增*
40. void create\_new\_record(Link\_Node\* list\_head) {
41. Link\_Node\* new\_node = (Link\_Node\*)malloc(sizeof(Link\_Node));
42. if (new\_node == NULL) {
43. puts("内存不足！");
44. return;
45. }
46. printf("输入学号: ");
47. scanf("%s", new\_node->record.stu\_id);
48. flush\_stdin();
49. *// 检查学号是否存在*
50. Link\_Node\* checker = list\_head->p\_next;
51. while (checker) {
52. if (strcmp(checker->record.stu\_id, new\_node->record.stu\_id) == 0) {
53. puts("错误：此学号已存在！");
54. free(new\_node);
55. return;
56. }
57. checker = checker->p\_next;
58. }
59. printf("输入姓名: ");
60. scanf("%s", new\_node->record.stu\_name);
61. flush\_stdin();
62. printf("输入性别: ");
63. scanf("%s", new\_node->record.stu\_gender);
64. flush\_stdin();
65. printf("输入年龄: ");
66. scanf("%d", &new\_node->record.stu\_age);
67. flush\_stdin();
68. printf("输入班级: ");
69. scanf("%s", new\_node->record.stu\_class);
70. flush\_stdin();
71. new\_node->p\_next = NULL;
72. Link\_Node\* current = list\_head;
73. while (current->p\_next != NULL) {
74. current = current->p\_next;
75. }
76. current->p\_next = new\_node;
77. puts("\n新记录已添加。");
78. }
79. *// 查（子菜单）*
80. static void find\_by\_student\_id(Link\_Node\* list\_head) {
81. char id[20];
82. printf("输入要查找的学号: ");
83. scanf("%s", id);
84. flush\_stdin();
85. Link\_Node\* ptr = list\_head->p\_next;
86. while (ptr) {
87. if (strcmp(ptr->record.stu\_id, id) == 0) {
88. printf("\n找到记录:\n");
89. printf("学号:%s, 姓名:%s, 性别:%s, 年龄:%d, 班级:%s\n",
90. ptr->record.stu\_id, ptr->record.stu\_name, ptr->record.stu\_gender, ptr->record.stu\_age, ptr->record.stu\_class);
91. return;
92. }
93. ptr = ptr->p\_next;
94. }
95. printf("数据库中无此学号 %s 的记录。\n", id);
96. }
97. static void find\_by\_student\_name(Link\_Node\* list\_head) {
98. char name[50];
99. int found\_count = 0;
100. printf("输入要查找的姓名: ");
101. scanf("%s", name);
102. flush\_stdin();
103. Link\_Node\* ptr = list\_head->p\_next;
104. puts("\n查找结果:");
105. while (ptr) {
106. if (strcmp(ptr->record.stu\_name, name) == 0) {
107. printf("学号:%s, 姓名:%s, 性别:%s, 年龄:%d, 班级:%s\n",
108. ptr->record.stu\_id, ptr->record.stu\_name, ptr->record.stu\_gender, ptr->record.stu\_age, ptr->record.stu\_class);
109. found\_count++;
110. }
111. ptr = ptr->p\_next;
112. }
113. printf("共找到 %d 个名为 %s 的学生。\n", found\_count, name);
114. }
115. void find\_record\_menu(Link\_Node\* list\_head) {
116. int choice;
117. do {
118. puts("\n--- 查找记录 ---");
119. puts("  1. 按学号查找");
120. puts("  2. 按姓名查找");
121. puts("  0. 返回");
122. printf("你的选择: ");
123. scanf("%d", &choice);
124. flush\_stdin();
125. if (choice == 1) find\_by\_student\_id(list\_head);
126. else if (choice == 2) find\_by\_student\_name(list\_head);
127. else if (choice != 0) puts("输入错误。");
128. } while (choice != 0);
129. }
130. *// 改（子菜单）*
131. static void update\_record(Link\_Node\* list\_head) {
132. char id[20];
133. printf("输入待修改记录的学号: ");
134. scanf("%s", id);
135. flush\_stdin();
136. Link\_Node\* target = list\_head->p\_next;
137. while (target) {
138. if (strcmp(target->record.stu\_id, id) == 0) {
139. puts("找到记录，请输入新数据。");
140. printf("输入新姓名: "); scanf("%s", target->record.stu\_name); flush\_stdin();
141. printf("输入新性别: "); scanf("%s", target->record.stu\_gender); flush\_stdin();
142. printf("输入新年龄: "); scanf("%d", &target->record.stu\_age); flush\_stdin();
143. printf("输入新班级: "); scanf("%s", target->record.stu\_class); flush\_stdin();
144. puts("\n记录更新完毕。");
145. return;
146. }
147. target = target->p\_next;
148. }
149. printf("数据库中无此学号 %s 的记录。\n", id);
150. }
151. *// 删*
152. static void remove\_record(Link\_Node\* list\_head) {
153. char id[20];
154. printf("输入待删除记录的学号: ");
155. scanf("%s", id);
156. flush\_stdin();
157. Link\_Node\* prev = list\_head;
158. Link\_Node\* curr = list\_head->p\_next;
159. while (curr) {
160. if (strcmp(curr->record.stu\_id, id) == 0) {
161. prev->p\_next = curr->p\_next;
162. free(curr);
163. printf("学号为 %s 的记录已被移除。\n", id);
164. return;
165. }
166. prev = curr;
167. curr = curr->p\_next;
168. }
169. printf("数据库中无此学号 %s 的记录。\n", id);
170. }
171. void alter\_record\_menu(Link\_Node\* list\_head) {
172. int choice;
173. do {
174. puts("\n--- 更改记录 ---");
175. puts("  1. 修改记录");
176. puts("  2. 移除记录");
177. puts("  0. 返回");
178. printf("你的选择: ");
179. scanf("%d", &choice);
180. flush\_stdin();
181. if (choice == 1) { update\_record(list\_head); persist\_records\_to\_disk(list\_head, "student\_data.db"); }
182. else if (choice == 2) { remove\_record(list\_head); persist\_records\_to\_disk(list\_head, "student\_data.db"); }
183. else if (choice != 0) puts("输入错误。");
184. } while (choice != 0);
185. }
186. *// 统计*
187. static void summarize\_by\_class(Link\_Node\* list\_head) {
188. char class\_name[50];
189. int total = 0;
190. printf("输入要统计的班级名: ");
191. scanf("%s", class\_name);
192. flush\_stdin();
193. for (Link\_Node\* p = list\_head->p\_next; p != NULL; p = p->p\_next) {
194. if (strcmp(p->record.stu\_class, class\_name) == 0) {
195. total++;
196. }
197. }
198. printf("班级 [%s] 的学生人数为: %d\n", class\_name, total);
199. }
200. static void summarize\_by\_gender(Link\_Node\* list\_head) {
201. int males = 0, females = 0;
202. for (Link\_Node\* p = list\_head->p\_next; p != NULL; p = p->p\_next) {
203. if (strcmp(p->record.stu\_gender, "男") == 0) males++;
204. else if (strcmp(p->record.stu\_gender, "女") == 0) females++;
205. }
206. printf("当前记录中，男生: %d 人, 女生: %d 人。\n", males, females);
207. }
208. void summarize\_data\_menu(Link\_Node\* list\_head) {
209. int choice;
210. do {
211. puts("\n--- 数据汇总 ---");
212. puts("  1. 按班级汇总");
213. puts("  2. 按性别汇总");
214. puts("  0. 返回");
215. printf("你的选择: ");
216. scanf("%d", &choice);
217. flush\_stdin();
218. if (choice == 1) summarize\_by\_class(list\_head);
219. else if (choice == 2) summarize\_by\_gender(list\_head);
220. else if (choice != 0) puts("输入错误。");
221. } while (choice != 0);
222. }
223. *// 显示全部*
224. void print\_all\_records(Link\_Node\* list\_head) {
225. if (!list\_head->p\_next) {
226. puts("当前没有任何记录。");
227. return;
228. }
229. puts("\n=============== 全部学生记录 ================");
230. printf("%-12s | %-12s | %-8s | %-8s | %-12s\n", "学号", "姓名", "性别", "年龄", "班级");
231. puts("-------------------------------------------------");
232. for (Link\_Node\* p = list\_head->p\_next; p != NULL; p = p->p\_next) {
233. printf("%-12s | %-12s | %-8s | %-8d | %-12s\n",
234. p->record.stu\_id, p->record.stu\_name, p->record.stu\_gender, p->record.stu\_age, p->record.stu\_class);
235. }
236. puts("=================================================");
237. }
238. *// 文件IO*
239. void persist\_records\_to\_disk(Link\_Node\* list\_head, const char\* filepath) {
240. FILE\* file\_ptr = fopen(filepath, "w");
241. if (!file\_ptr) {
242. printf("致命错误: 无法写入到文件 %s\n", filepath);
243. return;
244. }
245. for (Link\_Node\* p = list\_head->p\_next; p; p = p->p\_next) {
246. fprintf(file\_ptr, "%s %s %s %d %s\n",
247. p->record.stu\_id, p->record.stu\_name, p->record.stu\_gender, p->record.stu\_age, p->record.stu\_class);
248. }
249. fclose(file\_ptr);
250. }
251. Link\_Node\* load\_records\_from\_disk(const char\* filepath) {
252. Link\_Node\* head = (Link\_Node\*)malloc(sizeof(Link\_Node));
253. if (!head) exit(1);
254. head->p\_next = NULL;
255. FILE\* file\_ptr = fopen(filepath, "r");
256. if (!file\_ptr) {
257. printf("数据文件 %s 未找到, 系统将以空记录启动。\n", filepath);
258. return head;
259. }
260. Info\_Student buffer;
261. Link\_Node\* tail = head;
262. while (fscanf(file\_ptr, "%s %s %s %d %s", buffer.stu\_id, buffer.stu\_name, buffer.stu\_gender, &buffer.stu\_age, buffer.stu\_class) == 5) {
263. Link\_Node\* new\_node = (Link\_Node\*)malloc(sizeof(Link\_Node));
264. if (!new\_node) {
265. puts("加载时内存分配失败!");
266. break;
267. }
268. new\_node->record = buffer;
269. new\_node->p\_next = NULL;
270. tail->p\_next = new\_node;
271. tail = new\_node;
272. }
273. fclose(file\_ptr);
274. printf("已从 %s 成功加载记录。\n", filepath);
275. return head;
276. }
277. *// 释放内存*
278. void release\_all\_nodes(Link\_Node\* list\_head) {
279. Link\_Node\* p = list\_head;
280. while (p) {
281. Link\_Node\* to\_delete = p;
282. p = p->p\_next;
283. free(to\_delete);
284. }
285. }
286. #define \_CRT\_SECURE\_NO\_WARNINGS
287. #include "record\_manager.h"
288. #include <stdio.h>
289. #include <stdlib.h>
290. #define DB\_FILE "student\_data.txt"
291. void display\_main\_interface() {
292. puts("\n###### 学生档案管理系统 ######");
293. puts("   [1] 新增学生档案");
294. puts("   [2] 查找学生档案");
295. puts("   [3] 更改学生档案");
296. puts("   [4] 汇总档案数据");
297. puts("   [5] 显示所有档案");
298. puts("   [0] 退出系统");
299. puts("##############################");
300. printf("请输入操作代码 [0-5]: ");
301. }
302. int main(void) {
303. Link\_Node\* database\_head = load\_records\_from\_disk(DB\_FILE);
304. int command;
305. do {
306. display\_main\_interface();
307. scanf("%d", &command);
308. while (getchar() != '\n'); *// 清理*
309. switch (command) {
310. case 1:
311. create\_new\_record(database\_head);
312. persist\_records\_to\_disk(database\_head, DB\_FILE); *// 实时保存*
313. break;
314. case 2:
315. find\_record\_menu(database\_head);
316. break;
317. case 3:
318. alter\_record\_menu(database\_head);
319. *// 保存操作在子菜单中完成*
320. break;
321. case 4:
322. summarize\_data\_menu(database\_head);
323. break;
324. case 5:
325. print\_all\_records(database\_head);
326. break;
327. case 0:
328. puts("系统正在退出...");
329. break;
330. default:
331. puts("无效命令，请重新输入。");
332. break;
333. }
334. if (command != 0) {
335. puts("\n按回车键以继续...");
336. getchar();
337. system("cls"); *// Windows/DOS a specific command*
338. }
339. } while (command != 0);
340. release\_all\_nodes(database\_head);
341. puts("感谢使用！");
342. return 0;
343. }