

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

本系统的设计核心在于将数据操作分为清晰的内存处理和磁盘持久化两个层次，并通过一个主控模块来调度。系统采用C语言实现，以单向链表作为内存中的核心数据结构，以文本文件作为数据持久化的载体。

#### **总体架构与数据模型**

系统的静态结构和运行流程设计清晰。在数据模型方面，定义了 StudentEntry 结构体用于承载学生的具体信息（如学号、姓名等），以及 ListNode 结构体作为链表节点，每个节点包含一个 StudentEntry 数据域和一个指向下一节点的 next\_node 指针。系统采用带哨兵（Sentinel）头结点的单链表来组织所有 ListNode，该头结点不存储实际数据，旨在简化链表操作的边界条件处理。

程序的生命周期由 main 函数统一管理，遵循“**启动时加载，退出时保存**”的核心策略。

**加载**：程序启动时，main 函数首先调用 load\_entries\_from\_file 函数，从磁盘文件（students.txt）读取全部数据，并在内存中重建链表结构。

**操作**：随后，程序进入一个菜单驱动的 do-while 循环，用户在此期间执行的增、删、改、查等所有操作，都只针对内存中的链表进行，不涉及实时文件写入。

**保存**：只有当用户选择“退出”指令时，main 函数才会调用 save\_entries\_to\_file 函数，将内存中链表的最终状态一次性完整地覆写回磁盘文件。

**释放**：在程序终止前，调用 deallocate\_list 函数清理所有动态分配的节点内存，防止内存泄漏。

#### **核心数据操作算法**

所有在内存中对链表进行处理的核心算法均封装在 data\_manager.c 文件中。

**数据注入（增、改）**：

**新增条目 (insert\_new\_entry)**：算法首先动态分配一个 ListNode 节点。为保证数据完整性，在接收用户输入的学号后，会立即遍历链表进行唯一性校验。若学号已存在，则释放节点并中止操作。校验通过后，接收完整的学生信息，并通过遍历找到链表尾部，将新节点追加上去。

**编辑条目 (edit\_entry)**：该算法本质上是“查找并替换”。它首先根据用户提供的学号线性扫描链表以定位目标节点，找到后，直接用用户输入的新数据覆写该节点 content 域内的信息。

**数据移除与检索（删、查）**：

**删除条目 (delete\_entry)**：此算法采用经典的“前驱-当前”双指针法。prev 和 curr 两个指针同步后移遍历链表。当 curr 指向待删除节点时，通过 prev->next\_node = curr->next\_node 的指针操作将其从链中断开，随后调用 free 释放 curr 节点的内存。

**检索条目 (search\_by\_entry\_id, search\_by\_entry\_name)**：检索功能完全基于线性扫描算法。从链表的第一个有效数据节点开始逐一比对，按学号查找在找到首个匹配项后即返回，而按姓名查找则会遍历整个链表以统计所有同名记录。

#### **数据持久化与资源管理**

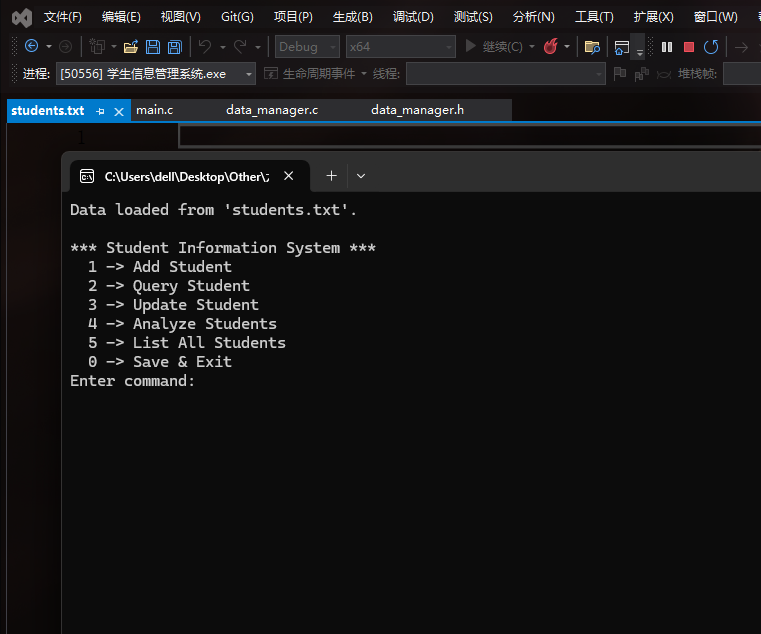
本系统的数据持久化和内存管理是其稳定运行的关键保障。

**数据序列化（保存）**：save\_entries\_to\_file 函数负责将内存中的链表数据序列化为文本格式。它以“写入”模式 ("w") 打开文件，此操作会清空文件原有内容。随后，函数遍历链表，将每个节点的 StudentEntry 信息通过 fprintf 格式化为一条由空格分隔的文本行，写入文件。

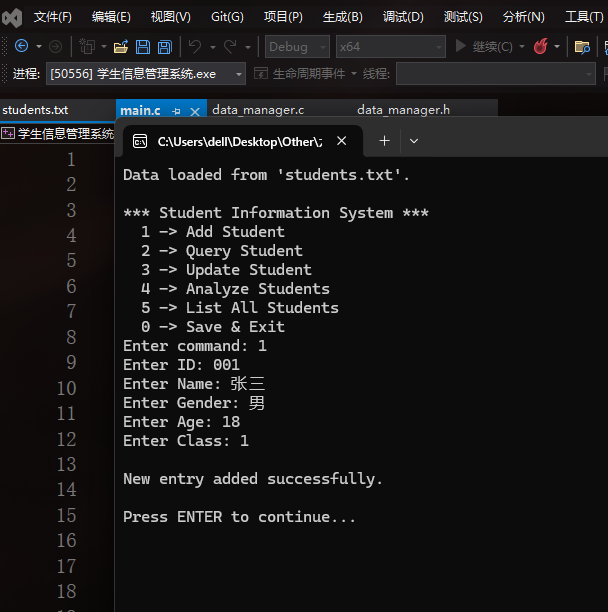
**数据反序列化（加载）**：load\_entries\_from\_file 函数负责在程序启动时从文件恢复数据。它以“读取”模式 ("r") 打开文件，若文件不存在则初始化一个空链表。若文件存在，则在 while 循环中通过 fscanf 逐条读取和解析数据。每解析成功一条，就动态分配一个新的 ListNode，并将其添加到链表尾部，从而在内存中重建数据结构。

**内存资源回收 (deallocate\_list)**：该函数是系统的“清道夫”。在程序退出前被调用，它会从头结点开始，遍历整个链表，并使用一个临时指针安全地释放每一个被 malloc 分配出来的节点，确保程序干净地将所有动态申请的内存归还给操作系统。

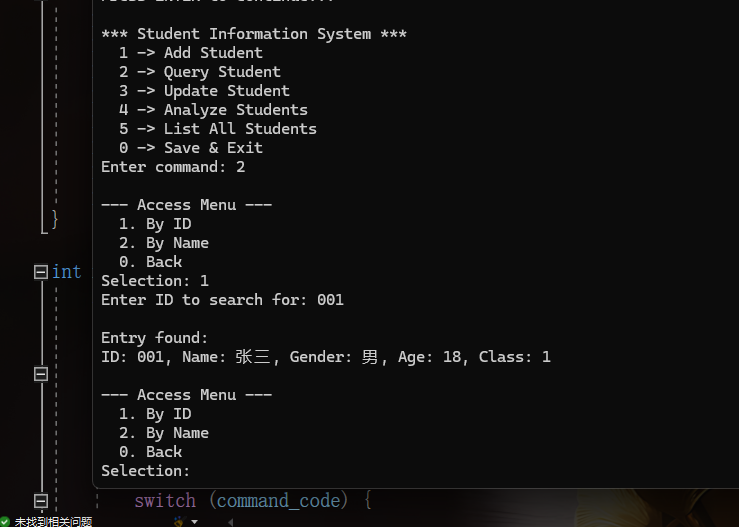
1. 算法实现
2. 程序主页面，启动与加载：程序启动后，首先显示从 students.txt 文件加载数据的提示，然后展示主菜单，等待用户操作 。



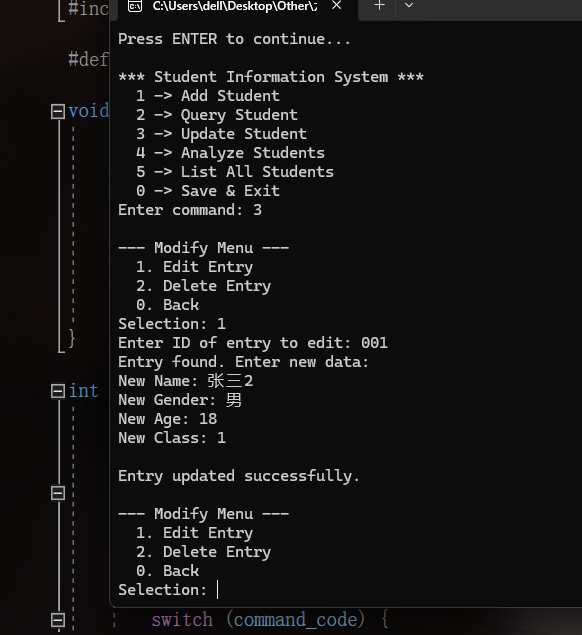
1. 录入学生信息，信息录入：下图展示了录入一个学号为"001"的学生信息的完整过程。添加成功后，系统提示数据已保存到文件 。



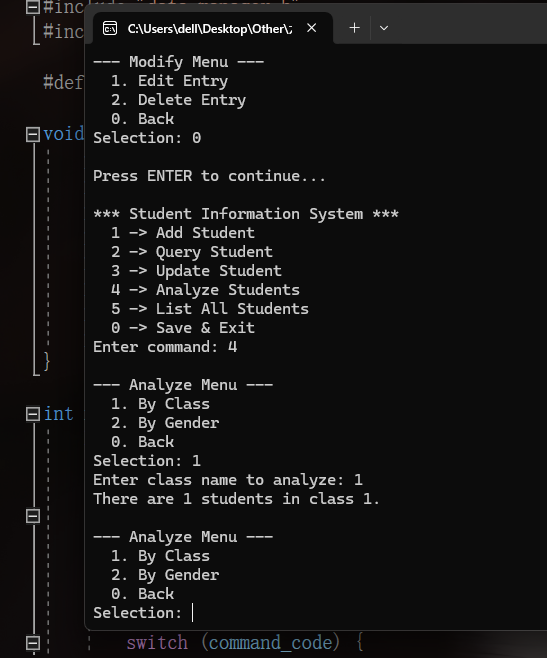
1. 查询学生信息，信息查询：系统提供了按学号和姓名两种查询方式。下图演示了分别使用这两种方式查询名为“张三”的学生，并准确返回了结果 。



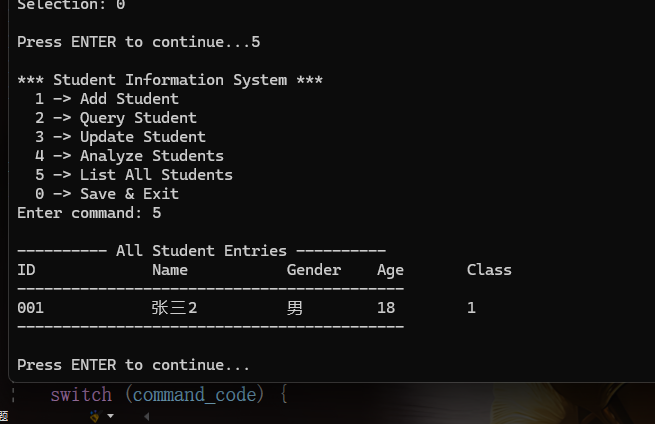
1. 更新学生信息，信息更新：用户可以修改或删除学生信息。下图展示了将学号"001"的学生的姓名从“张三”修改为“张三2”的操作。修改成功后，students.txt 文件中的内容也同步更新



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

本次学生信息管理系统的课程设计，对我而言，是一次将抽象的数据结构理论与具体的算法设计付诸实践的绝佳机会。它让我深刻认识到，程序的核心在于“数据结构+算法”。如何选择和设计合适的数据结构来组织信息，并在此基础上设计高效的算法来处理信息，是决定一个系统性能与优劣的根本。在这次实践中，我围绕单链表这一核心数据结构，展开了一系列算法的设计与优化，收获颇丰。

**一、 选择数据结构：为何是单链表？**

在项目设计之初，我面临第一个关键抉择：用什么来存储数量不定的学生信息？我考虑了两种方案：数组和链表。

**静态数组**：优点是实现简单，基于索引的访问速度快（O(1)）。但其致命缺点是大小固定。如果预设的数组过小，会导致数据无法录入；如果过大，则会造成严重的内存浪费。对于学生数量动态变化的管理系统而言，这显然不是一个理想的选择。

**动态数组**：虽然可以动态扩容，但扩容操作（重新分配内存、复制旧数据、释放旧内存）的开销较大，且在数据中间进行插入或删除操作时，需要移动大量元素，时间复杂度为O(n)。

**单链表**：它的最大优势在于其天生的动态性。每个节点在需要时才通过malloc创建，内存使用“按需分配”，完美地适应了数据量的动态变化。更重要的是，节点的插入和删除操作，理论上只需要修改相邻节点的指针即可，时间复杂度为O(1)​（不考虑查找目标节点的时间）。

综合权衡，考虑到本系统“动态增删”的核心需求，单链表成为了最契合的数据结构。这一选择过程，让我第一次亲身体会到在实际问题中进行技术选型和利弊分析（Trade-off）的重要性。

**二、 核心算法设计与实现**

确定了单链表作为基石后，我开始围绕它设计各项功能的算法。

**插入算法（尾插法）**：为了保证新录入的学生总是被添加到列表末尾，我采用了“尾插法”。算法的核心是设置一个while循环，从头结点开始，不断地将指针后移（p = p->next），直到找到最后一个节点（其next指针为NULL）。然后，将这个尾节点的next指向新创建的节点。这个算法虽然直观，但其时间复杂度为O(n)，因为每次插入都需要从头遍历。这也让我思考，如果系统对插入性能要求极高，或许可以增加一个指向链表尾部的tail指针，从而将插入操作优化为O(1)。

**删除算法（双指针法）**：删除是链表操作中的一个难点。要删除一个节点curr，我们必须知道它的前一个节点prev，以便执行prev->next = curr->next。为此，我设计了“双指针协同遍历”的算法。prev指针和curr指针同步向后移动，prev始终落后curr一步。当curr指向的目标节点被找到时，prev正好位于其前一个位置，此时执行删除操作就变得水到渠成。这个算法的精妙之处在于它巧妙地解决了“单向”链表无法回溯的问题。

**查询算法（线性扫描）**：无论是按学号查询还是按姓名查询，其底层算法都是对链表的线性扫描（Linear Scan）。从头结点开始，逐一检查每个节点的数据域是否与目标值匹配。按学号查询是唯一的，找到后即可break。而按姓名查询可能存在重名，因此必须遍历完整个链表。我清楚地认识到，这种查询方式的效率是其短板，时间复杂度为O(n)。这也引发了我的进一步思考：如果数据量巨大，如何优化查询？如果用学号作为主键，或许可以采用哈希表（用空间换时间）或者平衡二叉搜索树（如AVL树或红黑树）来组织数据，从而将查询效率提升至$O(1)$或$O(\log n)$。

**三、 算法与数据持久化的结合**

文件操作的引入，为算法设计带来了新的维度——序列化与反序列化。

**保存（序列化）**：saveToFile函数本质上是一个序列化算法。它将内存中结构化的Node链表，转换成可以存储在磁盘上的、线性的、基于文本的格式。我选择了简单的空格分隔格式，fprintf函数在此处扮演了核心的格式化角色。

**读取（反序列化）**：loadFromFile函数则是反序列化算法。它逐行读取文本文件，使用fscanf将线性文本流解析回结构化的Student数据，并在此基础上动态重建内存中的链表结构。调试这一过程让我深刻理解了数据在不同媒介（内存与磁盘）之间表示形式的转换。

总而言之，本次课程设计是一次名副其实的“数据结构与算法”之旅。我不再是停留在纸面上理解链表的增删改查，而是在一个真实的需求场景中，亲手设计、实现并评估了这些算法。我学会了根据问题特性选择合适的数据结构，分析算法的时间复杂度，并思考其优化的可能性。这种将理论知识应用于解决实际问题的能力，我相信，是计算机科学学习中最核心、最宝贵的收获。

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 DATA\_MANAGER\_H
3. #define DATA\_MANAGER\_H
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. *// 学生条目*
8. typedef struct {
9. char entry\_id[20];
10. char entry\_name[50];
11. char entry\_gender[10];
12. int entry\_age;
13. char entry\_class[50];
14. } StudentEntry;
15. *// 链表节点*
16. typedef struct ListNode {
17. StudentEntry content;
18. struct ListNode\* next\_node;
19. } ListNode;
20. *// --- 接口函数声明 ---*
21. *// 数据持久化*
22. ListNode\* load\_entries\_from\_file(const char\* file\_path);
23. void save\_entries\_to\_file(ListNode\* list\_head, const char\* file\_path);
24. *// 功能接口*
25. void insert\_new\_entry(ListNode\* list\_head);
26. void access\_entry\_submenu(ListNode\* list\_head);
27. void modify\_entry\_submenu(ListNode\* list\_head);
28. void analyze\_data\_submenu(ListNode\* list\_head);
29. void list\_all\_entries(ListNode\* list\_head);
30. *// 内存管理*
31. void deallocate\_list(ListNode\* list\_head);
32. #endif *// DATA\_MANAGER\_H*
33. #define \_CRT\_SECURE\_NO\_WARNINGS
34. #include "data\_manager.h"
35. *// 内部辅助函数：清空标准输入*
36. static void clean\_input\_stream() {
37. while (getchar() != '\n');
38. }
39. *// 增*
40. void insert\_new\_entry(ListNode\* list\_head) {
41. ListNode\* new\_node = (ListNode\*)malloc(sizeof(ListNode));
42. if (!new\_node) {
43. puts("Error: Memory allocation failed.");
44. return;
45. }
46. printf("Enter ID: ");
47. scanf("%s", new\_node->content.entry\_id);
48. clean\_input\_stream();
49. *// 唯一性检查*
50. for (ListNode\* p = list\_head->next\_node; p; p = p->next\_node) {
51. if (strcmp(p->content.entry\_id, new\_node->content.entry\_id) == 0) {
52. puts("Error: This ID already exists.");
53. free(new\_node);
54. return;
55. }
56. }
57. printf("Enter Name: "); scanf("%s", new\_node->content.entry\_name); clean\_input\_stream();
58. printf("Enter Gender: "); scanf("%s", new\_node->content.entry\_gender); clean\_input\_stream();
59. printf("Enter Age: "); scanf("%d", &new\_node->content.entry\_age); clean\_input\_stream();
60. printf("Enter Class: "); scanf("%s", new\_node->content.entry\_class); clean\_input\_stream();
61. new\_node->next\_node = NULL;
62. ListNode\* walker = list\_head;
63. while (walker->next\_node) {
64. walker = walker->next\_node;
65. }
66. walker->next\_node = new\_node;
67. puts("\nNew entry added successfully.");
68. }
69. *// 查*
70. static void search\_by\_entry\_id(ListNode\* list\_head) {
71. char id[20];
72. printf("Enter ID to search for: ");
73. scanf("%s", id);
74. clean\_input\_stream();
75. for (ListNode\* p = list\_head->next\_node; p; p = p->next\_node) {
76. if (strcmp(p->content.entry\_id, id) == 0) {
77. printf("\nEntry found:\nID: %s, Name: %s, Gender: %s, Age: %d, Class: %s\n",
78. p->content.entry\_id, p->content.entry\_name, p->content.entry\_gender, p->content.entry\_age, p->content.entry\_class);
79. return;
80. }
81. }
82. printf("No entry found with ID %s.\n", id);
83. }
84. static void search\_by\_entry\_name(ListNode\* list\_head) {
85. char name[50];
86. int results = 0;
87. printf("Enter Name to search for: ");
88. scanf("%s", name);
89. clean\_input\_stream();
90. puts("\nSearch Results:");
91. for (ListNode\* p = list\_head->next\_node; p; p = p->next\_node) {
92. if (strcmp(p->content.entry\_name, name) == 0) {
93. printf("ID: %s, Name: %s, Gender: %s, Age: %d, Class: %s\n",
94. p->content.entry\_id, p->content.entry\_name, p->content.entry\_gender, p->content.entry\_age, p->content.entry\_class);
95. results++;
96. }
97. }
98. printf("Found %d record(s).\n", results);
99. }
100. void access\_entry\_submenu(ListNode\* list\_head) {
101. int sel;
102. do {
103. puts("\n--- Access Menu ---\n  1. By ID\n  2. By Name\n  0. Back");
104. printf("Selection: ");
105. scanf("%d", &sel);
106. clean\_input\_stream();
107. if (sel == 1) search\_by\_entry\_id(list\_head);
108. else if (sel == 2) search\_by\_entry\_name(list\_head);
109. } while (sel != 0);
110. }
111. *// 改*
112. static void edit\_entry(ListNode\* list\_head) {
113. char id[20];
114. printf("Enter ID of entry to edit: ");
115. scanf("%s", id);
116. clean\_input\_stream();
117. for (ListNode\* p = list\_head->next\_node; p; p = p->next\_node) {
118. if (strcmp(p->content.entry\_id, id) == 0) {
119. puts("Entry found. Enter new data:");
120. printf("New Name: "); scanf("%s", p->content.entry\_name); clean\_input\_stream();
121. printf("New Gender: "); scanf("%s", p->content.entry\_gender); clean\_input\_stream();
122. printf("New Age: "); scanf("%d", &p->content.entry\_age); clean\_input\_stream();
123. printf("New Class: "); scanf("%s", p->content.entry\_class); clean\_input\_stream();
124. puts("\nEntry updated successfully.");
125. return;
126. }
127. }
128. printf("No entry found with ID %s.\n", id);
129. }
130. *// 删*
131. static void delete\_entry(ListNode\* list\_head) {
132. char id[20];
133. printf("Enter ID of entry to delete: ");
134. scanf("%s", id);
135. clean\_input\_stream();
136. ListNode\* prev = list\_head;
137. ListNode\* curr = list\_head->next\_node;
138. while (curr) {
139. if (strcmp(curr->content.entry\_id, id) == 0) {
140. prev->next\_node = curr->next\_node;
141. free(curr);
142. printf("Entry with ID %s has been deleted.\n", id);
143. return;
144. }
145. prev = curr;
146. curr = curr->next\_node;
147. }
148. printf("No entry found with ID %s.\n", id);
149. }
150. void modify\_entry\_submenu(ListNode\* list\_head) {
151. int sel;
152. do {
153. puts("\n--- Modify Menu ---\n  1. Edit Entry\n  2. Delete Entry\n  0. Back");
154. printf("Selection: ");
155. scanf("%d", &sel);
156. clean\_input\_stream();
157. if (sel == 1) edit\_entry(list\_head);
158. else if (sel == 2) delete\_entry(list\_head);
159. } while (sel != 0);
160. }
161. *// 统计*
162. static void analyze\_by\_class(ListNode\* list\_head) {
163. char class\_id[50];
164. int num = 0;
165. printf("Enter class name to analyze: ");
166. scanf("%s", class\_id);
167. clean\_input\_stream();
168. for (ListNode\* p = list\_head->next\_node; p; p = p->next\_node) {
169. if (strcmp(p->content.entry\_class, class\_id) == 0) num++;
170. }
171. printf("There are %d students in class %s.\n", num, class\_id);
172. }
173. static void analyze\_by\_gender(ListNode\* list\_head) {
174. int males = 0, females = 0;
175. for (ListNode\* p = list\_head->next\_node; p; p = p->next\_node) {
176. if (strcmp(p->content.entry\_gender, "男") == 0) males++;
177. else if (strcmp(p->content.entry\_gender, "女") == 0) females++;
178. }
179. printf("Gender distribution: %d male(s), %d female(s).\n", males, females);
180. }
181. void analyze\_data\_submenu(ListNode\* list\_head) {
182. int sel;
183. do {
184. puts("\n--- Analyze Menu ---\n  1. By Class\n  2. By Gender\n  0. Back");
185. printf("Selection: ");
186. scanf("%d", &sel);
187. clean\_input\_stream();
188. if (sel == 1) analyze\_by\_class(list\_head);
189. else if (sel == 2) analyze\_by\_gender(list\_head);
190. } while (sel != 0);
191. }
192. *// 显示全部*
193. void list\_all\_entries(ListNode\* list\_head) {
194. if (!list\_head->next\_node) {
195. puts("No entries to display.");
196. return;
197. }
198. puts("\n---------- All Student Entries ----------");
199. printf("%-15s%-15s%-10s%-10s%-15s\n", "ID", "Name", "Gender", "Age", "Class");
200. puts("-------------------------------------------");
201. for (ListNode\* p = list\_head->next\_node; p; p = p->next\_node) {
202. printf("%-15s%-15s%-10s%-10d%-15s\n", p->content.entry\_id, p->content.entry\_name, p->content.entry\_gender, p->content.entry\_age, p->content.entry\_class);
203. }
204. puts("-------------------------------------------");
205. }
206. *// 文件IO*
207. void save\_entries\_to\_file(ListNode\* list\_head, const char\* file\_path) {
208. FILE\* f\_out = fopen(file\_path, "w");
209. if (!f\_out) {
210. printf("Critical Error: Cannot open %s for writing.\n", file\_path);
211. return;
212. }
213. for (ListNode\* p = list\_head->next\_node; p; p = p->next\_node) {
214. fprintf(f\_out, "%s %s %s %d %s\n", p->content.entry\_id, p->content.entry\_name, p->content.entry\_gender, p->content.entry\_age, p->content.entry\_class);
215. }
216. fclose(f\_out);
217. }
218. ListNode\* load\_entries\_from\_file(const char\* file\_path) {
219. ListNode\* head = (ListNode\*)malloc(sizeof(ListNode));
220. if (!head) exit(1);
221. head->next\_node = NULL;
222. FILE\* f\_in = fopen(file\_path, "r");
223. if (!f\_in) {
224. printf("Data file '%s' not found. Starting with an empty database.\n", file\_path);
225. return head;
226. }
227. StudentEntry temp;
228. while (fscanf(f\_in, "%s %s %s %d %s", temp.entry\_id, temp.entry\_name, temp.entry\_gender, &temp.entry\_age, temp.entry\_class) != EOF) {
229. ListNode\* new\_node = (ListNode\*)malloc(sizeof(ListNode));
230. if (!new\_node) continue;
231. new\_node->content = temp;
232. new\_node->next\_node = NULL;
233. ListNode\* tail = head;
234. while (tail->next\_node) {
235. tail = tail->next\_node;
236. }
237. tail->next\_node = new\_node;
238. }
239. fclose(f\_in);
240. printf("Data loaded from '%s'.\n", file\_path);
241. return head;
242. }
243. void deallocate\_list(ListNode\* list\_head) {
244. ListNode\* current = list\_head;
245. while (current) {
246. ListNode\* next = current->next\_node;
247. free(current);
248. current = next;
249. }
250. }
251. #define \_CRT\_SECURE\_NO\_WARNINGS
252. #include "data\_manager.h"
253. #include <stdio.h>
254. #define DATA\_SOURCE "students.txt"
255. void print\_main\_menu() {
256. puts("\n\*\*\* Student Information System \*\*\*");
257. puts("  1 -> Add Student");
258. puts("  2 -> Query Student");
259. puts("  3 -> Update Student");
260. puts("  4 -> Analyze Students");
261. puts("  5 -> List All Students");
262. puts("  0 -> Save & Exit");
263. printf("Enter command: ");
264. }
265. int main(int argc, char\* argv[]) {
266. ListNode\* data\_head = load\_entries\_from\_file(DATA\_SOURCE);
267. int command\_code;
268. do {
269. print\_main\_menu();
270. scanf("%d", &command\_code);
271. while (getchar() != '\n'); *// flush*
272. switch (command\_code) {
273. case 1:
274. insert\_new\_entry(data\_head);
275. break;
276. case 2:
277. access\_entry\_submenu(data\_head);
278. break;
279. case 3:
280. modify\_entry\_submenu(data\_head);
281. break;
282. case 4:
283. analyze\_data\_submenu(data\_head);
284. break;
285. case 5:
286. list\_all\_entries(data\_head);
287. break;
288. case 0:
289. save\_entries\_to\_file(data\_head, DATA\_SOURCE);
290. puts("Data saved. Exiting...");
291. break;
292. default:
293. puts("Invalid command code.");
294. }
295. if (command\_code != 0) {
296. printf("\nPress ENTER to continue...");
297. getchar();
298. }
299. } while (command\_code != 0);
300. deallocate\_list(data\_head);
301. return 0;
302. }