

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

本学生信息管理系统采用C语言开发，其核心设计思想是“内存计算与文件持久化分离”。系统在运行时，所有学生数据均被加载到内存的单向链表中进行高效操作；在程序退出时，内存中的数据被统一写回磁盘文件，以实现数据的持久化存储。

#### **1. 数据结构设计**

为了有效地组织学生信息，系统设计了两种核心的结构体，并在头文件

student\_management.h 中定义 。

**struct Student**: 这是一个基础数据结构，用于封装单个学生的具体信息。它包含了学号（id）、姓名（name）、性别（gender）、年龄（age）和班级（className）等多个异构数据成员 。

**struct Node**: 这是构成链表的基本单元（节点）。每个Node结构体包含两个部分：Student类型的data域，用于存放具体的学生信息 。一个指向下一个节点的指针 struct Node\* next，它像链条一样将所有节点按逻辑顺序串联起来 。

本系统采用**带头结点的单链表**。这意味着链表中存在一个不存储实际学生数据的“哨兵”或“哑”（dummy）头结点。这个头结点的data域为空，其next指针指向第一个真正存储学生信息的节点。采用这种设计极大地简化了链表的操作，特别是在链表头部进行插入和删除时，无需对头结点进行特殊处理，使得算法逻辑更加统一。

#### **2. 程序生命周期与持久化算法**

程序的整体流程由

main.c中的main函数控制，其生命周期算法如下 ：

**初始化（加载）**：程序启动时，首先调用load\_from\_file()函数 。该函数尝试从指定的文本文件（students.txt）中读取数据，并在内存中动态地创建链表以还原数据状态。

**交互循环**：进入一个do-while循环，通过show\_main\_menu()向用户展示操作菜单 。

**事件分发**：使用switch语句根据用户的数字选择，调用student\_management.c中对应的功能函数（如add\_student, query\_student\_menu等）来执行具体操作 。

**保存与退出**：当用户选择0时，程序会调用save\_to\_file()函数，将内存中链表的当前所有数据一次性覆写到磁盘文件students.txt中 。之后，程序退出循环。

**内存释放**：在程序完全退出前，调用free\_list()函数，遍历并释放整个链表（包括头结点）占用的所有动态分配内存，防止内存泄漏 。

#### **3. 核心功能模块算法**

所有核心功能均在student\_management.c文件中实现 。

**输入模块 (add\_student)**

**内存分配**：使用malloc()为新学生创建一个新的Node节点 。

**数据录入**：提示用户依次输入学号、姓名等信息，并存入新节点的data域 。

**唯一性校验**：调用is\_id\_unique()辅助函数，遍历链表检查新输入的学号是否已存在。若存在，则释放新分配的节点并提示错误，终止添加 。

**尾部插入**：若学号唯一，则从头结点开始遍历链表，直到找到链表的最后一个节点（其next指针为NULL）。将该尾节点的next指针指向新创建的节点，完成插入 。

**查询模块 (query\_student\_menu)**

该模块提供了两种查询方式：按学号查询和按姓名查询 。

**按学号查询 (search\_by\_id)**: 这是一个线性搜索算法。从链表的第一个学生节点开始遍历，使用strcmp()比较学号。由于学号是唯一的，找到第一个匹配项后即可打印信息并立即返回 。

**按姓名查询 (search\_by\_name)**: 同样是线性搜索，但考虑到可能存在同名学生，算法会遍历整个链表，打印所有姓名匹配的学生信息，并统计匹配的总数 。

**更新模块 (update\_student\_menu)**

**修改学生信息 (modify\_student)**: 首先通过用户输入的学号进行线性搜索，定位到目标节点。若找到，则提示用户输入各项新信息，并直接覆盖该节点data域中的原有数据 。

**删除学生信息 (delete\_student)**: 这是一个经典的链表节点删除算法。使用两个指针prev和curr，prev初始化为头结点，curr初始化为第一个学生节点 。同步向后移动两个指针进行遍历，使得prev始终是curr的前驱节点。当curr指向的节点学号匹配时，执行prev->next = curr->next;，此操作将curr节点从链表中“跨过”，使其在逻辑上被断开 。最后，调用free(curr);释放被断开节点的内存空间，防止内存泄漏 。

**统计模块 (statistics\_menu)**

提供按班级和按性别两种统计方式 。其核心算法非常直接：初始化一个或多个计数器，然后完整地遍历一次链表。在遍历过程中，对每个节点的数据进行条件判断（如班级名或性别是否匹配），若符合条件则递增相应的计数器。遍历结束后，输出统计结果 。

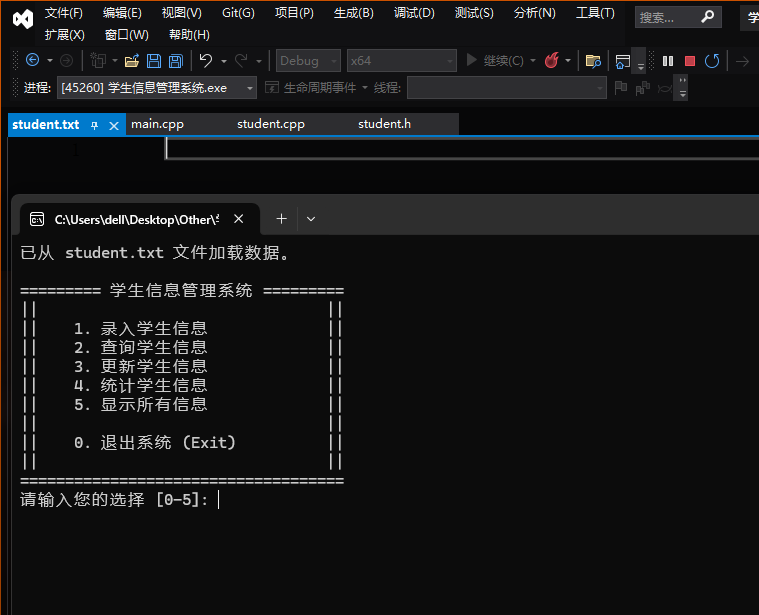
**文件I/O与内存管理**

**数据保存 (save\_to\_file)**: 此为**序列化**过程。函数以写入模式("w")打开文件，该模式会清空文件原有内容。随后遍历链表，使用fprintf()将每个节点的各项数据格式化为一行文本，用空格隔开，写入文件 。

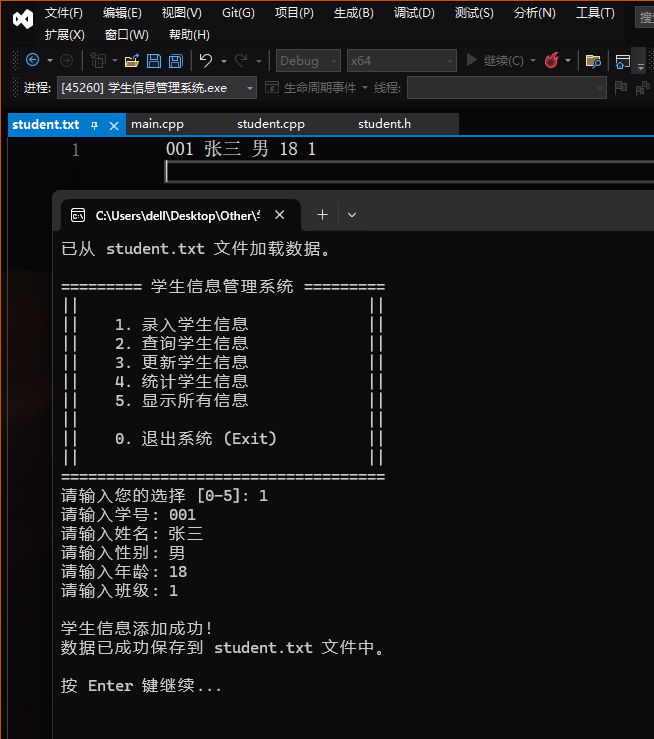
**数据加载 (load\_from\_file)**: 此为**反序列化**过程。函数以读取模式("r")打开文件。若文件不存在（如首次运行），则创建一个空的头结点并返回 。若文件存在，则在while循环中反复调用fscanf()，每次读取一行数据并解析到临时的Student结构体中。每成功读取一条记录，就为其malloc一个新节点，并采用尾插法将其加入到正在构建的链表中 。

**内存释放 (free\_list)**: 为确保程序结束时无内存泄漏，该函数从头结点开始，使用一个临时指针temp保存当前节点，然后将主指针移至下一个节点，最后free(temp)。此过程循环进行，直至整个链表被安全销毁

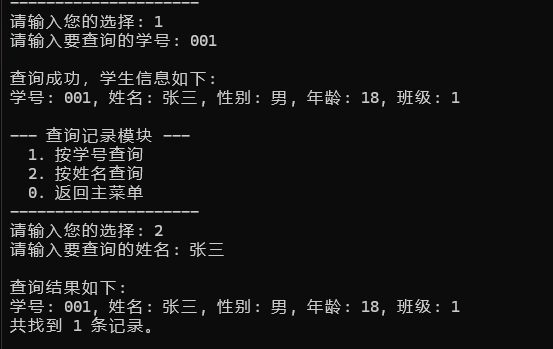
1. 算法实现
2. 程序主页面，启动与加载：程序启动后，首先显示从 student.txt 文件加载数据的提示，然后展示主菜单，等待用户操作 。



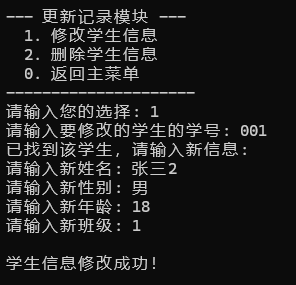
1. 录入学生信息，信息录入：下图展示了录入一个学号为"001"的学生信息的完整过程。添加成功后，系统提示数据已保存到文件 。

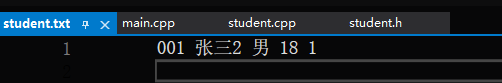


1. 查询学生信息，信息查询：系统提供了按学号和姓名两种查询方式。下图演示了分别使用这两种方式查询名为“张三”的学生，并准确返回了结果 。

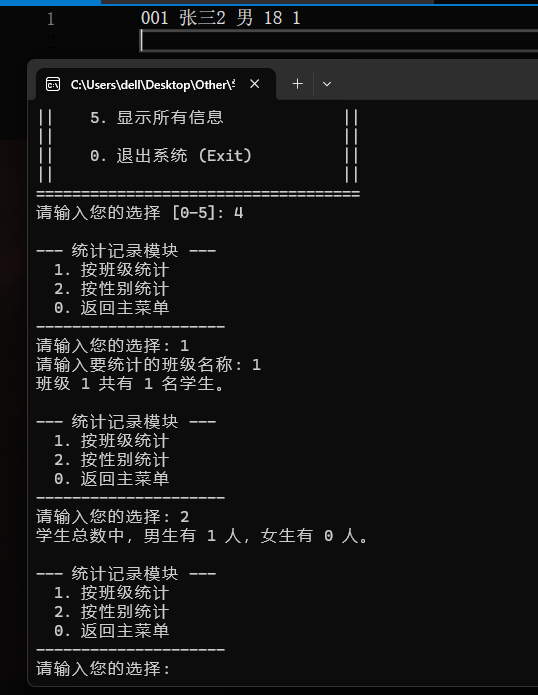


1. 更新学生信息，信息更新：用户可以修改或删除学生信息。下图展示了将学号"001"的学生的姓名从“张三”修改为“张三2”的操作。修改成功后，student.txt 文件中的内容也同步更新





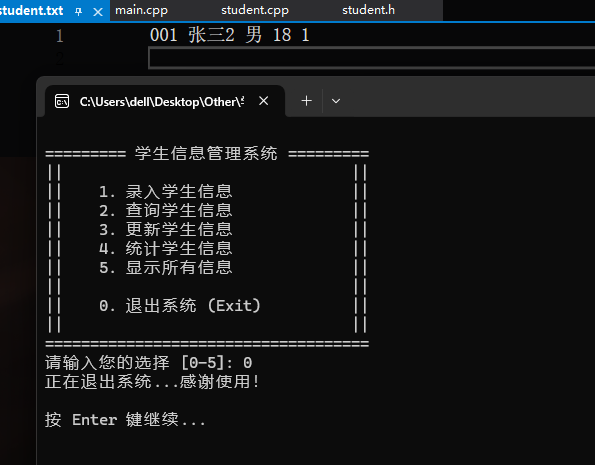
1. 统计学生信息，信息统计与显示：系统能够按班级和性别进行统计，也可以完整地列出所有学生的信息，方便用户全局浏览 。

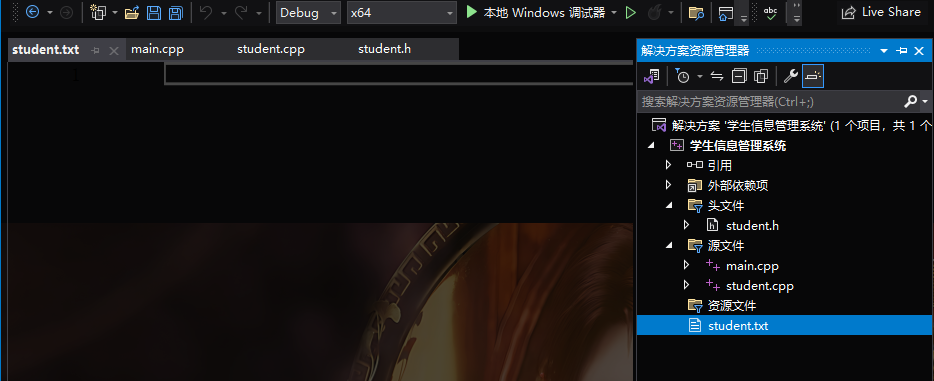


1. 显示所有信息



1. 退出系统 Exit





四．心得体会

本次C语言课程设计，我选择并完成了“学生信息管理系统”这一课题。这不仅仅是一次简单的课程任务，更是一场对C语言核心概念——尤其是指针与动态内存管理的深度探索与洗礼。在整个开发过程中，我从最初对理论的模糊认知，到最终能够熟练运用相关知识构建一个功能完备的系统，期间的收获与感悟，可谓刻骨铭心。我深刻体会到，C语言的精髓，正在于它赋予程序员的那份对内存最直接、最精细的掌控力。

**一、 征服指针：从心生畏惧到运用自如**

在项目初期，我面临的最大挑战无疑是指针。教科书上对指针的定义、\*与&的用法虽然早已背诵，但在实际应用中，它们如同捉摸不定的幽灵，常常导致程序崩溃或逻辑混乱。尤其是在构建链表时，Node\*、head->next、current->next = newNode这些操作，起初在我看来就像一团乱麻。我常常因为分不清一个变量是节点本身、是指向节点的指针，还是指向指针的指针而陷入困境，程序也因此频繁地报出“段错误（Segmentation Fault）”。

为了攻克这一难关，我采取了最原始也最有效的方法——画图。我准备了草稿纸和笔，每当要实现一个链表操作时，我都会在纸上画出内存的示意图。方框代表struct Node，里面再画小方框代表data和next指针。我用箭头表示指针的指向，一步一步地模拟代码的执行流程。例如，在实现“删除节点”功能时，我清晰地画出了prev、curr两个指针如何协同移动，当curr找到目标后，prev->next = curr->next这行代码如何像“断开铁链”一样，将curr节点从链表中逻辑上移除。紧接着，free(curr)这行代码又如何像“回收废品”一样，将这块被孤立的内存空间归还给系统。通过这种可视化的方式，抽象的指针操作变得具体而直观，我终于理解了指针的本质——它就是地址，是对内存空间的一种引用。这次经历让我明白，面对复杂的编程概念，借助工具将抽象问题具象化，是极其重要的学习方法。

**二、 malloc与free：程序员的责任与担当**

如果说指针是C语言的利剑，那么动态内存管理就是驾驭这把利剑的剑法。本次设计强制要求使用链表，这意味着我必须使用malloc函数在程序的“堆区”动态地申请内存来存放每一个新录入的学生信息。每一次addStudent函数的调用，本质上都是一次向操作系统的“借款”。我深刻认识到，有借必须有还，free函数就是我们的“还款”工具。

在开发过程中，我逐渐建立起了对内存泄漏的警惕性。我认识到，每一次malloc都必须有一个与之对应的free。在删除学生信息时，free掉被删除的节点是理所当然的。但更隐蔽的情况发生在程序退出的时刻。如果程序结束时不释放整个链表所占用的内存，虽然现代操作系统能够回收进程资源，但在大型、长期运行的程序中，这无疑是灾难性的。因此，我专门编写了freeList函数，在main函数退出前，通过循环遍历整个链表，逐一释放每一个节点。

此外，我还处理了一种更细微的内存泄漏场景：在addStudent函数中，当用户输入的学号与已有学号重复时，程序会提示错误并返回。此时，函数开头为newNode申请的内存如果没有被free掉，它就会成为一个无法被引用的“内存孤儿”，从而造成泄漏。我在错误处理路径上加上了free(newNode)，确保了程序的严谨性。这个小小的细节让我领悟到，一个优秀的C程序员，必须时刻将内存管理铭记于心，它不仅是一种技术，更是一种对系统资源负责的严М态度。

**三、 综合实践：知识的融会贯通**

除了指针和内存管理，本次课设也全面锻炼了我对C语言其他知识点的综合运用能力。结构体struct的定义让我学会了如何将不同类型的数据打包成一个有意义的整体；typedef关键字则让我的代码更具可读性；多文件编程（.h与.c）让我初步体验了模块化开发的思想；文件操作fopen、fprintf、fscanf等函数，则让我的程序第一次拥有了“记忆”，实现了数据的持久化存储，这让我兴奋不已。

总而言之，这次学生信息管理系统的课程设计，是一次理论与实践的完美结合。它强迫我直面C语言最核心、最困难但也最迷人的部分。通过亲手绘制内存图、小心翼翼地管理每一次malloc与free、细致地调试每一个功能模块，我不仅成功地完成了一个应用系统，更重要的是，我在这个过程中真正地“看见”了内存，“触摸”到了指针，深刻领悟了C语言设计的哲学与精髓。这段经历为我未来的编程学习之路，打下了坚不可摧的基石。

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 STUDENT\_MANAGEMENT\_H
3. #define STUDENT\_MANAGEMENT\_H
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. *// 学生信息结构体*
8. typedef struct Student {
9. char id[20];
10. char name[50];
11. char gender[10];
12. int age;
13. char className[50];
14. } Student;
15. *// 链表节点*
16. typedef struct Node {
17. Student data;
18. struct Node\* next;
19. } Node;
20. *// --- 函数原型 ---*
21. *// 加载数据*
22. Node\* load\_from\_file(const char\* filename);
23. *// 保存数据*
24. void save\_to\_file(Node\* head, const char\* filename);
25. *// 释放链表*
26. void free\_list(Node\* head);
27. *// 功能模块*
28. void add\_student(Node\* head);
29. void query\_student\_menu(Node\* head);
30. void update\_student\_menu(Node\* head);
31. void statistics\_menu(Node\* head);
32. void display\_all\_students(Node\* head);
33. #endif *// STUDENT\_MANAGEMENT\_H*
34. #define \_CRT\_SECURE\_NO\_WARNINGS
35. #include "student\_management.h"
36. *// 清理输入缓冲区*
37. static void clear\_buffer() {
38. int c;
39. while ((c = getchar()) != '\n' && c != EOF);
40. }
41. *// 检查学号唯一性*
42. static int is\_id\_unique(Node\* head, const char\* id) {
43. Node\* current = head->next;
44. while (current != NULL) {
45. if (strcmp(current->data.id, id) == 0) {
46. return 0; *// 不唯一*
47. }
48. current = current->next;
49. }
50. return 1; *// 唯一*
51. }
52. *// 1. 录入学生*
53. void add\_student(Node\* head) {
54. Node\* new\_node = (Node\*)malloc(sizeof(Node));
55. if (!new\_node) {
56. printf("内存分配失败！\n");
57. return;
58. }
59. printf("请输入学号: ");
60. scanf("%s", new\_node->data.id);
61. clear\_buffer();
62. if (!is\_id\_unique(head, new\_node->data.id)) {
63. printf("错误: 该学号已经存在！\n");
64. free(new\_node);
65. return;
66. }
67. printf("请输入姓名: ");
68. scanf("%s", new\_node->data.name);
69. clear\_buffer();
70. printf("请输入性别: ");
71. scanf("%s", new\_node->data.gender);
72. clear\_buffer();
73. printf("请输入年龄: ");
74. scanf("%d", &new\_node->data.age);
75. clear\_buffer();
76. printf("请输入班级: ");
77. scanf("%s", new\_node->data.className);
78. clear\_buffer();
79. new\_node->next = NULL;
80. Node\* tail = head;
81. while (tail->next != NULL) {
82. tail = tail->next;
83. }
84. tail->next = new\_node;
85. printf("\n学生信息添加成功！\n");
86. }
87. *// 2. 查询 (按ID)*
88. static void search\_by\_id(Node\* head) {
89. char id[20];
90. printf("请输入要查询的学号: ");
91. scanf("%s", id);
92. clear\_buffer();
93. Node\* current = head->next;
94. while (current != NULL) {
95. if (strcmp(current->data.id, id) == 0) {
96. printf("\n查询成功，信息如下:\n");
97. printf("学号: %s, 姓名: %s, 性别: %s, 年龄: %d, 班级: %s\n",
98. current->data.id, current->data.name, current->data.gender, current->data.age, current->data.className);
99. return;
100. }
101. current = current->next;
102. }
103. printf("未找到学号为 %s 的学生。\n", id);
104. }
105. *// 2. 查询 (按姓名)*
106. static void search\_by\_name(Node\* head) {
107. char name[50];
108. int count = 0;
109. printf("请输入要查询的姓名: ");
110. scanf("%s", name);
111. clear\_buffer();
112. Node\* current = head->next;
113. printf("\n查询结果:\n");
114. while (current != NULL) {
115. if (strcmp(current->data.name, name) == 0) {
116. printf("学号: %s, 姓名: %s, 性别: %s, 年龄: %d, 班级: %s\n",
117. current->data.id, current->data.name, current->data.gender, current->data.age, current->data.className);
118. count++;
119. }
120. current = current->next;
121. }
122. if (count == 0) {
123. printf("未找到姓名为 %s 的学生。\n", name);
124. }
125. else {
126. printf("共找到 %d 条记录。\n", count);
127. }
128. }
129. void query\_student\_menu(Node\* head) {
130. int option;
131. do {
132. printf("\n--- 查询菜单 ---\n");
133. printf("  1. 按学号查询\n");
134. printf("  2. 按姓名查询\n");
135. printf("  0. 返回上级\n");
136. printf("------------------\n");
137. printf("请输入选项: ");
138. scanf("%d", &option);
139. clear\_buffer();
140. switch (option) {
141. case 1: search\_by\_id(head); break;
142. case 2: search\_by\_name(head); break;
143. case 0: break;
144. default: printf("无效选项!\n"); break;
145. }
146. } while (option != 0);
147. }
148. *// 3. 修改*
149. static void modify\_student(Node\* head) {
150. char id[20];
151. printf("请输入要修改的学生的学号: ");
152. scanf("%s", id);
153. clear\_buffer();
154. Node\* current = head->next;
155. while (current) {
156. if (strcmp(current->data.id, id) == 0) {
157. printf("找到学生，请输入新信息:\n");
158. printf("请输入新姓名: ");
159. scanf("%s", current->data.name);
160. clear\_buffer();
161. printf("请输入新性别: ");
162. scanf("%s", current->data.gender);
163. clear\_buffer();
164. printf("请输入新年龄: ");
165. scanf("%d", &current->data.age);
166. clear\_buffer();
167. printf("请输入新班级: ");
168. scanf("%s", current->data.className);
169. clear\_buffer();
170. printf("\n信息修改成功！\n");
171. return;
172. }
173. current = current->next;
174. }
175. printf("未找到该学号的学生。\n");
176. }
177. *// 3. 删除*
178. static void delete\_student(Node\* head) {
179. char id[20];
180. printf("请输入要删除的学生的学号: ");
181. scanf("%s", id);
182. clear\_buffer();
183. Node\* prev = head;
184. Node\* curr = head->next;
185. while (curr) {
186. if (strcmp(curr->data.id, id) == 0) {
187. prev->next = curr->next;
188. free(curr);
189. printf("学号为 %s 的学生已删除。\n", id);
190. return;
191. }
192. prev = curr;
193. curr = curr->next;
194. }
195. printf("未找到该学号的学生。\n");
196. }
197. void update\_student\_menu(Node\* head) {
198. int option;
199. do {
200. printf("\n--- 更新菜单 ---\n");
201. printf("  1. 修改学生信息\n");
202. printf("  2. 删除学生信息\n");
203. printf("  0. 返回上级\n");
204. printf("------------------\n");
205. printf("请输入选项: ");
206. scanf("%d", &option);
207. clear\_buffer();
208. switch (option) {
209. case 1: modify\_student(head); break;
210. case 2: delete\_student(head); break;
211. case 0: break;
212. default: printf("无效选项!\n"); break;
213. }
214. } while (option != 0);
215. }
216. *// 4. 统计 (按班级)*
217. static void stats\_by\_class(Node\* head) {
218. char class\_name[50];
219. int count = 0;
220. printf("请输入要统计的班级: ");
221. scanf("%s", class\_name);
222. clear\_buffer();
223. Node\* current = head->next;
224. while (current) {
225. if (strcmp(current->data.className, class\_name) == 0) {
226. count++;
227. }
228. current = current->next;
229. }
230. printf("班级 %s 共有 %d 名学生。\n", class\_name, count);
231. }
232. *// 4. 统计 (按性别)*
233. static void stats\_by\_gender(Node\* head) {
234. int male = 0, female = 0;
235. Node\* current = head->next;
236. while (current) {
237. if (strcmp(current->data.gender, "男") == 0) {
238. male++;
239. }
240. else if (strcmp(current->data.gender, "女") == 0) {
241. female++;
242. }
243. current = current->next;
244. }
245. printf("学生总数中，男生 %d 人，女生 %d 人。\n", male, female);
246. }
247. void statistics\_menu(Node\* head) {
248. int option;
249. do {
250. printf("\n--- 统计菜单 ---\n");
251. printf("  1. 按班级统计\n");
252. printf("  2. 按性别统计\n");
253. printf("  0. 返回上级\n");
254. printf("------------------\n");
255. printf("请输入选项: ");
256. scanf("%d", &option);
257. clear\_buffer();
258. switch (option) {
259. case 1: stats\_by\_class(head); break;
260. case 2: stats\_by\_gender(head); break;
261. case 0: break;
262. default: printf("无效选项!\n"); break;
263. }
264. } while (option != 0);
265. }
266. *// 5. 显示所有*
267. void display\_all\_students(Node\* head) {
268. if (head->next == NULL) {
269. printf("系统中没有学生信息。\n");
270. return;
271. }
272. printf("\n======================= 所有学生信息 =======================\n");
273. printf("%-15s %-15s %-10s %-10s %-15s\n", "学号", "姓名", "性别", "年龄", "班级");
274. printf("------------------------------------------------------------\n");
275. Node\* current = head->next;
276. while (current != NULL) {
277. printf("%-15s %-15s %-10s %-10d %-15s\n",
278. current->data.id, current->data.name, current->data.gender, current->data.age, current->data.className);
279. current = current->next;
280. }
281. printf("============================================================\n");
282. }
283. *// 6. 文件操作*
284. void save\_to\_file(Node\* head, const char\* filename) {
285. FILE\* fp = fopen(filename, "w");
286. if (!fp) {
287. printf("错误: 无法打开文件 %s 进行写入。\n", filename);
288. return;
289. }
290. Node\* current = head->next;
291. while (current) {
292. fprintf(fp, "%s %s %s %d %s\n",
293. current->data.id, current->data.name, current->data.gender, current->data.age, current->data.className);
294. current = current->next;
295. }
296. fclose(fp);
297. printf("数据已保存到 %s\n", filename);
298. }
299. Node\* load\_from\_file(const char\* filename) {
300. Node\* head = (Node\*)malloc(sizeof(Node));
301. if (!head) {
302. printf("关键内存分配失败！\n");
303. exit(1);
304. }
305. head->next = NULL;
306. FILE\* fp = fopen(filename, "r");
307. if (!fp) {
308. printf("未找到数据文件 %s, 将创建新系统。\n", filename);
309. return head;
310. }
311. Student temp\_stu;
312. while (fscanf(fp, "%s %s %s %d %s", temp\_stu.id, temp\_stu.name, temp\_stu.gender, &temp\_stu.age, temp\_stu.className) != EOF) {
313. Node\* new\_node = (Node\*)malloc(sizeof(Node));
314. if (!new\_node) {
315. printf("加载数据时内存分配失败！\n");
316. break;
317. }
318. new\_node->data = temp\_stu;
319. new\_node->next = NULL;
320. Node\* tail = head;
321. while (tail->next) {
322. tail = tail->next;
323. }
324. tail->next = new\_node;
325. }
326. fclose(fp);
327. printf("已从 %s 文件加载数据。\n", filename);
328. return head;
329. }
330. *// 释放链表*
331. void free\_list(Node\* head) {
332. Node\* current = head;
333. while (current != NULL) {
334. Node\* temp = current;
335. current = current->next;
336. free(temp);
337. }
338. }
339. #define \_CRT\_SECURE\_NO\_WARNINGS
340. #include "student\_management.h"
341. #include <stdio.h>
342. #define DATA\_FILE "students.txt"
343. void show\_main\_menu() {
344. printf("\n========= 学生信息管理系统 =========\n");
345. printf("||                                ||\n");
346. printf("||    1. 录入学生信息             ||\n");
347. printf("||    2. 查询学生信息             ||\n");
348. printf("||    3. 更新学生信息             ||\n");
349. printf("||    4. 统计学生信息             ||\n");
350. printf("||    5. 显示所有信息             ||\n");
351. printf("||                                ||\n");
352. printf("||    0. 保存并退出               ||\n");
353. printf("||                                ||\n");
354. printf("====================================\n");
355. printf("请输入您的选择 [0-5]: ");
356. }
357. int main() {
358. Node\* student\_list\_head = load\_from\_file(DATA\_FILE);
359. int user\_choice;
360. do {
361. show\_main\_menu();
362. scanf("%d", &user\_choice);
363. while (getchar() != '\n'); *// 清理缓冲区*
364. switch (user\_choice) {
365. case 1:
366. add\_student(student\_list\_head);
367. break;
368. case 2:
369. query\_student\_menu(student\_list\_head);
370. break;
371. case 3:
372. update\_student\_menu(student\_list\_head);
373. break;
374. case 4:
375. statistics\_menu(student\_list\_head);
376. break;
377. case 5:
378. display\_all\_students(student\_list\_head);
379. break;
380. case 0:
381. save\_to\_file(student\_list\_head, DATA\_FILE);
382. printf("正在退出系统...感谢使用！\n");
383. break;
384. default:
385. printf("无效的输入，请输入 0-5 之间的数字。\n");
386. break;
387. }
388. if (user\_choice != 0) {
389. printf("\n按 Enter 键返回主菜单...");
390. getchar();
391. }
392. } while (user\_choice != 0);
393. free\_list(student\_list\_head);
394. return 0;
395. }

样式导出