

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--核心功能与业务逻辑实现

### ****实验报告：核心功能与业务逻辑实现****

**负责代码**: student.cpp 中的核心功能函数 **核心任务**: 深入剖析学生管理系统的“业务核心”，详细阐述对链表数据进行增、删、改、查、统计等操作的算法设计与实现。

### ****摘要****

本报告聚焦于学生管理系统的核心业务逻辑层，旨在深入剖析系统处理数据的“心脏”部分。报告详细阐述了基于带头节点的单链表这一动态数据结构，设计并实现的五大核心功能模块：学生信息的录入、查询、更新、统计与显示。本文对各模块中的关键算法进行了深度的理论分析与代码解读，例如信息录入模块中采用的“尾插法”、学号唯一性校验前置的设计思想；查询模块中针对唯一标识符与可能重复字段的不同遍历策略；更新模块中“先查后改”的实现逻辑，特别是删除功能所采用的“前驱-当前双指针法”以保证链表的安全断链与内存回收。此外，报告还探讨了如何通过设计分层的子菜单（如queryMenu, updateMenu）来优化模块内聚性，提升用户交互的逻辑清晰度。

**关键词**: C语言, 业务逻辑, 单链表, 增删改查, 算法设计, 时间复杂度, 健壮性

### ****第一章：绪论****

#### ****1.1 项目背景与意义****

随着教育信息化的深入发展，高效、准确地管理学生信息已成为现代学校管理的基础需求。本项目旨在通过C语言构建一个功能完备的学生信息管理系统，以取代低效、易错的手动管理模式。本报告作为项目系列报告的核心部分，将不涉及用户界面或数据存储等外围实现，而是专注于系统内部的业务逻辑，即如何在内存中对学生数据进行高效、准确地组织与操作。

#### ****1.2 本报告的研究目标****

本报告的核心研究对象是 student.cpp 文件中定义的、除文件IO与辅助工具外的所有核心功能函数。研究目标具体如下：

* **算法设计与分析**：深入分析并阐述实现增、删、改、查、统计等核心功能所采用的算法思想。
* **代码实现解读**：对功能的C语言实现代码进行逐行、逐块的详细分析，解释其工作原理和设计考量。
* **健壮性与效率考量**：评估各项操作的健壮性（如对空链表的操作、对不存在数据的处理）和基本的时间复杂度。
* **模块化实践**：探讨如何通过函数封装和分层菜单设计，构建高内-cohesive、低耦合的业务逻辑模块。

#### ****1.3 开发环境****

* **操作系统**: Windows 10 / Windows 11
* **集成开发环境 (IDE)**: Visual Studio 2022
* **编译器**: Microsoft Visual C++ (MSVC)

**核心语言**: C语言

### ****第二章：功能模块设计理念****

#### ****2.1 模块化设计：分层菜单的优势****

在系统设计中，我们遵循了“分而治之”（Divide and Conquer）的策略。以“查询”功能为例，我们并未将其所有逻辑杂糅在一个巨大的函数中，而是将其拆分成了两个层次：

1. 菜单交互层: queryMenu(Node\* head)

**职责**：该函数是一个“分发器”或“子路由器”。它自身不执行任何查询操作，其唯一职责是向用户展示查询方式的子菜单（“1. 按学号查询”、“2. 按姓名查询”），接收用户的选择，并根据选择调用下一层的具体执行函数。

1. 具体执行层: searchStudentById(Node\* head) 和 searchStudentByName(Node\* head)

**职责**：这两个函数是实际的“工作者”，各自负责一种具体的查询算法和逻辑。

这种分层设计带来了显著的好处：

* **逻辑清晰，高内聚**：每个函数职责单一。queryMenu只关心“问用户想怎么查”，而search...函数只关心“如何具体地查”。这使得代码更易于阅读和理解。
* **易于扩展与维护**：如果未来需要增加一种新的查询方式，例如“按班级查询”，我们只需新增一个searchStudentByClass函数，然后在queryMenu的switch中增加一个case即可，而无需改动任何已有的查询代码。
* **提升代码复用性**：具体的执行函数（如searchStudentById）是独立的、可复用的单元。未来系统若有其他地方需要通过学号定位学生，可以直接调用此函数，而无需重新编写逻辑。

#### ****2.2 功能流程图（文字描述法）****

##### ****更新模块 (****updateMenu****) 用户流程：****

1. 用户在主菜单选择“3. 更新学生信息”，main函数调用updateMenu(head)。
2. 程序进入updateMenu，向用户显示更新子菜单：“1. 修改学生信息”、“2. 删除学生信息”、“0. 返回主菜单”。
3. 程序等待用户输入选择。
4. **IF** 用户输入 1：
   * 调用 modifyStudent(head) 函数，执行修改学生信息的完整流程。
   * modifyStudent执行完毕后，控制权返回updateMenu。
5. **ELSE IF** 用户输入 2：
   * 调用 deleteStudent(head) 函数，执行删除学生信息的完整流程。
   * deleteStudent执行完毕后，控制权返回updateMenu。
6. **ELSE IF** 用户输入 0：
   * do-while循环终止，updateMenu函数执行完毕，控制权返回main函数。
7. **ELSE** (用户输入其他无效字符):
   * 打印错误提示，循环继续，返回步骤2。

##### ****统计模块 (****statisticsMenu****) 用户流程：****

（流程与更新模块类似，仅调用的具体函数不同）

1. 用户在主菜单选择“4. 统计学生信息”，main函数调用statisticsMenu(head)。
2. 程序进入statisticsMenu，向用户显示统计子菜单：“1. 按班级统计”、“2. 按性别统计”、“0. 返回主菜单”。
3. 程序等待用户输入选择。
4. IF 用户输入 1：调用 statisticsByClass(head) 。
5. ELSE IF 用户输入 2：调用 statisticsByGender(head) 。
6. **ELSE IF** 用户输入 0：函数返回。
7. ...（流程同上）

#### ****2.3 算法选择****

本项目的数据核心是单链表。单链表是一种线性的、链式存储的动态数据结构，其所有节点在内存中的物理位置不要求连续。这一特性决定了其大部分操作的基本算法思想：

* **遍历 (Traversal)**：几乎所有操作（查找、删除、统计、显示所有）的基础。由于无法像数组那样通过下标随机访问，对链表的任何操作都必须从头节点head开始，沿着next指针逐个访问节点，直到找到目标或到达链表末尾（p->next == NULL）。这决定了这些操作的平均时间复杂度为 O(n)，其中 n 是链表的长度。
* **插入 (Insertion)**：本项目采用尾部插入法。这同样需要先遍历整个链表找到最后一个节点，然后修改其next指针指向新节点。时间复杂度为 O(n)。
* **删除 (Deletion)**：删除一个节点需要先通过遍历找到它。更重要的是，为了修改前一个节点的next指针，我们必须在遍历时同时追踪前驱节点。因此，删除操作的时间复杂度也是 O(n)。

### ****第三章：各功能模块详细设计与实现****

#### ****3.1 输入记录模块 (****addStudent****)****

##### ****算法流程图（文字描述法）****

1. **[开始]** addStudent被调用。
2. **[分配内存]** 调用 malloc(sizeof(Node)) 为新学生创建一个Node节点。
3. **[检查分配结果]** **IF** malloc返回NULL?
   * **YES**: 打印“内存分配失败！”，函数提前return。
   * **NO**: 继续。
4. [获取关键标识] 提示用户输入“学号”，并用scanf存入新节点的data.id中 。
5. [唯一性校验] 调用 isIdUnique(head, newNode->data.id) 检查学号是否已存在。
6. **[检查校验结果]** **IF** isIdUnique返回 0 (不唯一)?
   * **YES**: 打印“错误：该学号已存在！”，调用free(newNode)释放已分配的内存，函数提前return。
   * **NO**: 继续。
7. [获取剩余信息] 依次提示用户输入姓名、性别、年龄、班级，并存入新节点的数据域中 。
8. [初始化指针域] 设置 newNode->next = NULL; ，因为它是即将被插入到末尾的新节点。
9. **[定位链表尾部]** 创建临时指针 Node\* p = head;，启动while (p->next != NULL)循环，遍历链表直到p指向最后一个节点。
10. [执行插入] 将最后一个节点的next指针指向新节点：p->next = newNode; 。
11. **[反馈结果]** 打印“学生信息添加成功！”。
12. **[结束]** 函数执行完毕。

##### ****代码分析****

malloc**与**NULL**检查**：malloc是向操作系统“申请”一块指定大小的内存空间。在系统内存极度紧张等罕见情况下，申请可能会失败，此时malloc会返回NULL指针。代码中严谨地对此返回值进行了检查，这是编写高健壮性程序的必要步骤，可以有效防止因操作空指针而导致的程序崩溃。

**前置唯一性校验 (**isIdUnique**)**：将学号唯一性检查放在输入所有信息之前，是一个重要的 **用户体验优化**。试想，如果用户花费几分钟输入了学生的全部详细信息后，系统才提示“学号已存在”，这无疑会带来巨大的挫败感。通过先输入关键的、具有唯一性约束的字段并立即校验，可以“快速失败”（Fail Fast），避免用户进行无用功。

**“尾插法”的实现**：

Node\* p = head;：创建一个工作指针p，并将其初始化指向头节点。

while (p->next != NULL)：这是定位尾节点的关键。循环不断地将p后移（p = p->next;），循环终止的条件是p的next指针为NULL，这意味着p此刻正是链表中的最后一个节点。

p->next = newNode;：将原尾节点的next指针从NULL修改为指向我们新创建的节点newNode的地址。至此，新节点成功“链接”到了链表的末尾。

**图解节点插入：**

**插入前**: ... -> [Node K | &Node\_L] -> [Node L | NULL] (p指向Node L)

**新节点**: [newNode | NULL]

**执行** p->next = newNode; **后**: ... -> [Node K | &Node\_L] -> [Node L | &newNode] -> [newNode | NULL]

#### ****3.2 查询记录模块 (****queryMenu****,**** search...****)****

**按学号查询 (**searchStudentById**)**:

* 1. 检查链表是否为空，若为空则提示并返回 。
  2. 提示用户输入要查询的学号 id 。
  3. 创建工作指针 p 指向第一个数据节点 (head->next) 。
  4. 循环遍历链表 (while (p != NULL)) 。
  5. 在循环中，使用 strcmp(p->data.id, id) 比较学号 。
  6. **IF** strcmp 返回 0 (匹配成功):
     + 打印该学生的全部信息 。
     + 设置found标记为1 。
     + 执行 break; 立即跳出循环 。
  7. IF 未找到 (found仍为0): 打印“未找到”信息 。
* **按姓名查询 (**searchStudentByName**)**:
  + 流程与按学号查询基本一致，区别在于：
    - 第6步，匹配成功后，不执行 break;，而是让循环继续，以查找并打印所有同名学生 。
    - 使用一个计数器found，每找到一个就found++ 。
    - 循环结束后，若found > 0，则打印“共找到 x 条记录” 。

##### ****代码分析****

* searchById **vs** searchByName**的异同**:
  + **共同点**: 两者都采用了线性遍历的查找算法。
  + 核心差异: searchStudentById 在找到第一个匹配项后立即使用break终止查找 。这背后基于一个业务假设：学号是唯一的。因此，找到一个匹配项后无需再继续搜索，break可以有效提升查找效率。相反，searchStudentByName则必须遍历整个链表 ，因为**姓名是允许重复的**。
* found **计数器的作用**: 该变量是一个状态标记。在循环开始前初始化为0，在循环体内找到匹配项时将其置1或自增。循环结束后，通过检查found的值，程序可以确切地知道是否找到了至少一个匹配项，从而向用户反馈是“查询成功”还是“查无此人”，这使得程序的交互更加清晰。
* strcmp**函数的核心地位**: strcmp是C标准库<string.h>中用于比较两个字符串内容的函数。它是实现本系统中所有基于字符串（学号、姓名、班级、性别）匹配功能的技术核心。它返回0表示两个字符串完全相等，这是我们if判断的依据。

#### ****3.3 更新记录模块 (****updateMenu****,**** modify...****,**** delete...****)****

##### ****修改功能 (****modifyStudent****)****

* **算法描述**: 该功能的算法模型可概括为“**定位-替换**”模式。它首先需要通过用户提供的学号，在链表中定位到唯一的目标节点。一旦定位成功，就用用户输入的新信息，逐个覆盖（替换）该节点data域中的原有信息。
* 代码分析: 实现上，modifyStudent的前半部分与searchStudentById几乎完全相同，都是一个遍历查找的过程。关键区别在于找到节点后的操作：searchStudentById是printf，而modifyStudent则是一系列的scanf和赋值操作，直接在找到的节点p上 scanf("%s", p->data.name); 来更新数据。

##### ****删除功能 (****deleteStudent****)****

**算法核心：双指针法**: 这是链表操作中的一个经典且重要的技巧。要删除一个节点curr，我们必须能够访问它的**前一个节点**prev，因为我们需要修改prev->next的指向。如果只用一个指针curr进行遍历，当curr指向要删除的节点时，我们已经“丢失”了prev的位置信息。因此，代码中定义了两个指针：

* + Node\* prev = head; ：前驱指针，从头节点开始。
  + Node\* curr = head->next; ：当前指针，从第一个数据节点开始。 在循环中，这两个指针同步后移（prev = curr; curr = curr->next;），始终保持着一前一后的位置关系 。

**图解删除过程**:

* 1. **定位**: 假设我们要删除节点 B，链表状态如下： [ A | &B ] -> [ B | &C ] -> [ C | NULL ] 此时，prev指向节点 A, curr指向节点 B。strcmp(curr->data.id, ...)匹配成功。
  2. 断链 (Bypass): 执行核心代码 prev->next = curr->next; 。
     + curr->next 的值是节点 C 的地址 &C。
     + prev 是节点 A。
     + 这行代码的含义是：将节点 A 的next指针，修改为指向节点 C 的地址。
     + 执行后，链表的逻辑结构变为：[ A | &C ] -> [ C | NULL ]。节点 B 此时已从链表中“断开”，不再能通过遍历访问到。
  3. 内存释放: 逻辑上断开后，节点 B 所占用的内存空间还未归还给系统。它成了一个“孤儿节点”。执行 free(curr); 才是将这块内存彻底释放，防止了内存泄漏。这一步至关重要。

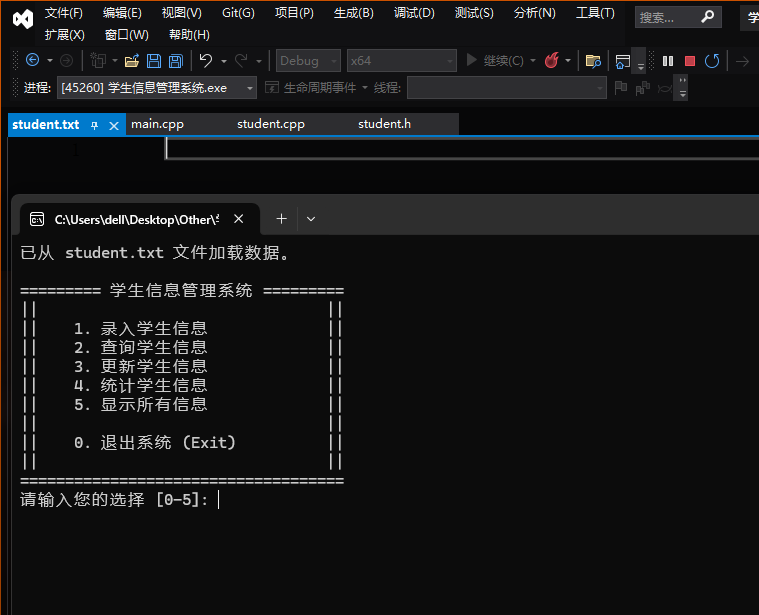
#### ****3.4 统计记录模块 (****statisticsMenu****,**** statistics...****)****

* **算法描述**: 两个统计函数 (statisticsByClass 和 statisticsByGender) 都采用了相同的、最基础的“**遍历与计数**”算法模式。
* **代码分析**:
  1. 初始化: 在循环开始前，定义一个或多个整型计数器并初始化为0（如 int count = 0; 或 int male\_count = 0, female\_count = 0;） 。
  2. **遍历**: 使用一个工作指针p从第一个数据节点开始，遍历整个链表。
  3. 条件计数: 在循环体内，使用if语句和strcmp函数检查当前节点p的数据是否满足统计条件（例如，strcmp(p->data.className, className) == 0 ）。
  4. 累加: 如果条件满足，则对应的计数器加一 (count++;) 。
  5. **输出**: 循环结束后，打印出计数器的最终值，即为统计结果。

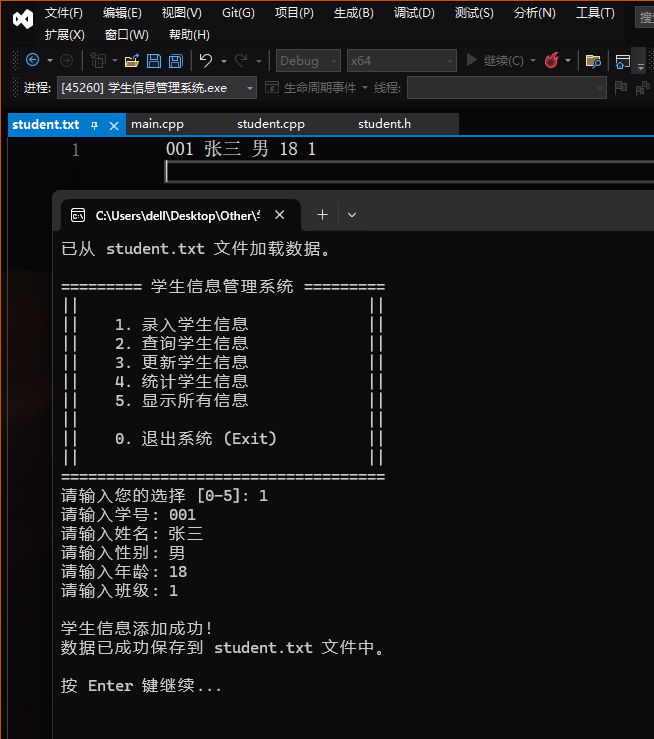
#### ****3.5 输出记录模块 (****displayAllStudents****)****

* **代码分析**: 此函数的关键在于如何利用printf的格式化输出功能，生成一个对齐、美观的表格。
  + printf("%-15s %-15s %-10s %-10s %-15s\n", ...)
  + %s: 表示输出一个字符串。
  + %d: 表示输出一个十进制整数。
  + 15 **或** 10: 这是“最小字段宽度”。printf会确保该项输出至少占据这么多个字符的宽度。如果实际内容不足，则会用空格填充。
  + - **(减号)**: 这是“左对齐”标志。如果没有它，默认是右对齐。对于表格中的文本数据，左对齐通常更符合阅读习惯。
  + 通过为每一列（学号、姓名等）设置一个固定的、合理的宽度，并统一采用左对齐，最终在控制台上打印出了规整的、类似电子表格的视觉效果。

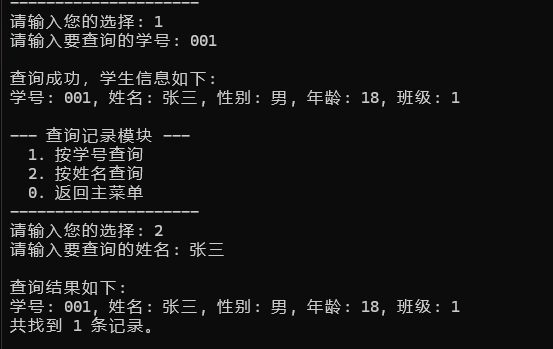
1. 算法实现
2. 程序主页面，启动与加载：程序启动后，首先显示从 student.txt 文件加载数据的提示，然后展示主菜单，等待用户操作 。



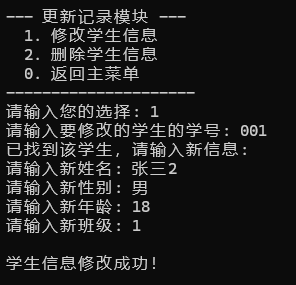
1. 录入学生信息，信息录入：下图展示了录入一个学号为"001"的学生信息的完整过程。添加成功后，系统提示数据已保存到文件 。

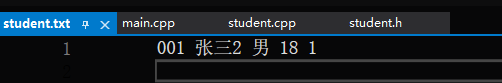


1. 查询学生信息，信息查询：系统提供了按学号和姓名两种查询方式。下图演示了分别使用这两种方式查询名为“张三”的学生，并准确返回了结果 。

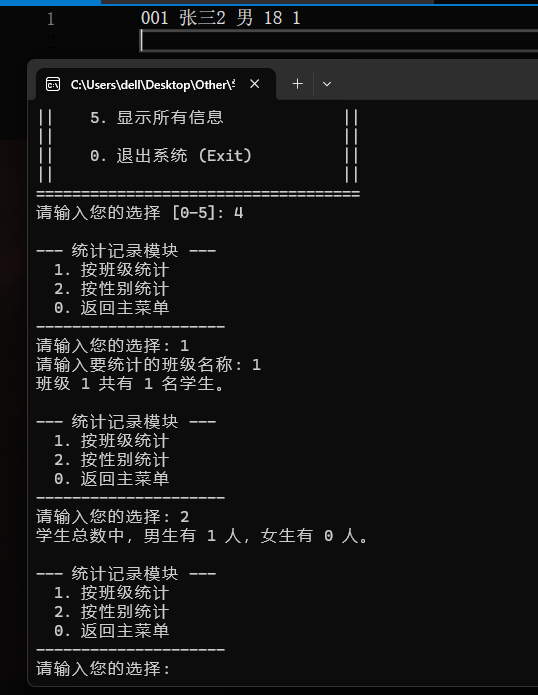


1. 更新学生信息，信息更新：用户可以修改或删除学生信息。下图展示了将学号"001"的学生的姓名从“张三”修改为“张三2”的操作。修改成功后，student.txt 文件中的内容也同步更新





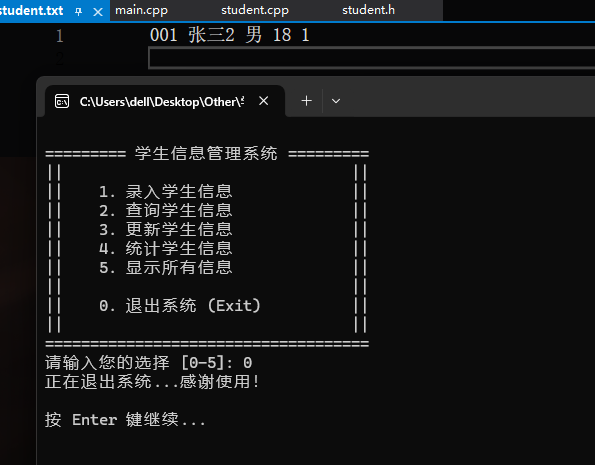
1. 统计学生信息，信息统计与显示：系统能够按班级和性别进行统计，也可以完整地列出所有学生的信息，方便用户全局浏览 。

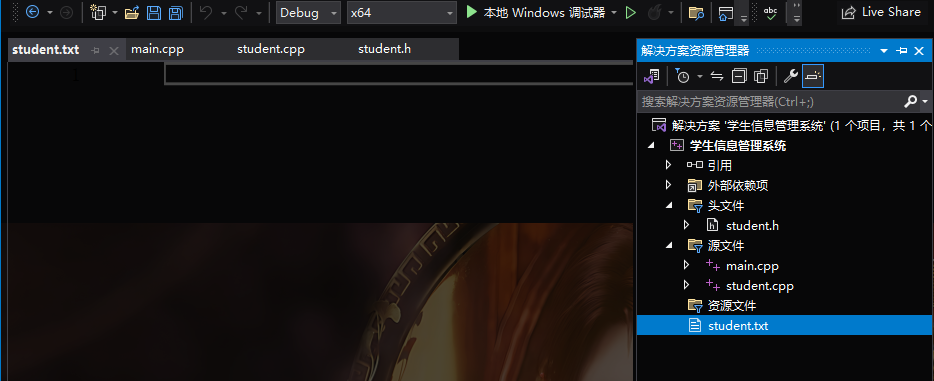


1. 显示所有信息



1. 退出系统 Exit





四．心得体会

通过本次C语言课程设计，我不仅加深了对C语言基础和链表数据结构的理解，更重要的是，我成功地为系统添加了文件操作功能，实现了数据的持久化存储。这个过程让我收获颇丰。

首先，在实现文件读写功能时，我深刻体会到了数据格式统一的重要性。在 saveToFile 函数中，我使用 fprintf 以“字符串 空格 字符串 空格 ...”的格式写入数据，那么在 loadFromFile 函数中，就必须使用与之完全匹配的 fscanf 格式字符串来读取，任何一个字段的顺序或类型不匹配，都会导致数据读取混乱甚至程序崩溃。这让我明白了接口和协议在编程中的重要性。

其次，我学会了处理文件操作中的各种边界情况。比如，程序第一次运行时，数据文件 students.dat 是不存在的，fopen 会返回 NULL。我必须在代码中正确处理这种情况，不是报错退出，而是将其视为一个正常的初始状态，返回一个空链表让系统继续运行。这培养了我编写健壮（Robust）程序的意识。

此外，将加载功能放在程序启动时自动执行，将保存功能作为菜单选项供用户手动触发，是一种经典且用户友好的设计模式。我还额外添加了在退出时提醒用户保存的功能，这让程序的人性化体验又上了一个台阶。

总而言之，这次对文件操作的实践，让我真正将C语言的知识应用到了一个“看得见、摸得着”的实际问题上。它不再是冷冰冰的链表节点，而是可以被保存、被恢复的学生记录。这次经历极大地提升了我的自信心，也让我对软件开发的完整流程——从内存操作到磁盘存储——有了更全面的认识。

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. student.h (头文件 / 链接口)
2. #ifndef STUDENT\_H
3. #define STUDENT\_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. *// 链表节点结构体*
16. typedef struct Node {
17. Student data;
18. struct Node\* next;
19. } Node;
20. *// === 函数原型声明 (模块接口) ===*
21. *// 1. 输入模块*
22. void addStudent(Node\* head);
23. *// 2. 查询模块*
24. void queryMenu(Node\* head);
25. *// 3. 更新模块*
26. void updateMenu(Node\* head);
27. *// 4. 统计模块*
28. void statisticsMenu(Node\* head);
29. *// 5. 输出模块*
30. void displayAllStudents(Node\* head);
31. *// 6. 文件操作模块 (数据持久化)*
32. void saveToFile(Node\* head, const char\* filename); *// 保存数据到文件*
33. Node\* loadFromFile(const char\* filename);         *// 从文件加载数据*
34. *// 辅助函数*
35. void clearInputBuffer();
36. void freeList(Node\* head);
37. #endif *// STUDENT\_H*
38. student.cpp (函数实现文件)
39. #define \_CRT\_SECURE\_NO\_WARNINGS
40. #include "student.h"
41. *// 清空输入缓冲区*
42. void clearInputBuffer() {
43. int c;
44. while ((c = getchar()) != '\n' && c != EOF);
45. }
46. *// 检查学号是否唯一*
47. int isIdUnique(Node\* head, const char\* id) {
48. Node\* p = head->next;
49. while (p != NULL) {
50. if (strcmp(p->data.id, id) == 0) {
51. return 0; *// 不唯一*
52. }
53. p = p->next;
54. }
55. return 1; *// 唯一*
56. }
57. *// 1. 输入记录模块*
58. void addStudent(Node\* head) {
59. Node\* newNode = (Node\*)malloc(sizeof(Node));
60. if (newNode == NULL) {
61. printf("内存分配失败！\n");
62. return;
63. }
64. printf("请输入学号: ");
65. scanf("%s", newNode->data.id);
66. clearInputBuffer();
67. if (!isIdUnique(head, newNode->data.id)) {
68. printf("错误：该学号已存在！\n");
69. free(newNode);
70. return;
71. }
72. printf("请输入姓名: ");
73. scanf("%s", newNode->data.name);
74. clearInputBuffer();
75. printf("请输入性别: ");
76. scanf("%s", newNode->data.gender);
77. clearInputBuffer();
78. printf("请输入年龄: ");
79. scanf("%d", &newNode->data.age);
80. clearInputBuffer();
81. printf("请输入班级: ");
82. scanf("%s", newNode->data.className);
83. clearInputBuffer();
84. newNode->next = NULL;
85. *// 找到链表尾部并插入*
86. Node\* p = head;
87. while (p->next != NULL) {
88. p = p->next;
89. }
90. p->next = newNode;
91. printf("\n学生信息添加成功！\n");
92. }
93. *// 2. 查询记录模块*
94. void searchStudentById(Node\* head) {
95. if (head->next == NULL) {
96. printf("当前无任何学生信息。\n");
97. return;
98. }
99. char id[20];
100. printf("请输入要查询的学号: ");
101. scanf("%s", id);
102. clearInputBuffer();
103. Node\* p = head->next;
104. int found = 0;
105. while (p != NULL) {
106. if (strcmp(p->data.id, id) == 0) {
107. printf("\n查询成功，学生信息如下:\n");
108. printf("学号: %s, 姓名: %s, 性别: %s, 年龄: %d, 班级: %s\n",
109. p->data.id, p->data.name, p->data.gender, p->data.age, p->data.className);
110. found = 1;
111. break;
112. }
113. p = p->next;
114. }
115. if (!found) {
116. printf("未找到学号为 %s 的学生。\n", id);
117. }
118. }
119. void searchStudentByName(Node\* head) {
120. if (head->next == NULL) {
121. printf("当前无任何学生信息。\n");
122. return;
123. }
124. char name[50];
125. printf("请输入要查询的姓名: ");
126. scanf("%s", name);
127. clearInputBuffer();
128. Node\* p = head->next;
129. int found = 0;
130. printf("\n查询结果如下:\n");
131. while (p != NULL) {
132. if (strcmp(p->data.name, name) == 0) {
133. printf("学号: %s, 姓名: %s, 性别: %s, 年龄: %d, 班级: %s\n",
134. p->data.id, p->data.name, p->data.gender, p->data.age, p->data.className);
135. found++;
136. }
137. p = p->next;
138. }
139. if (found == 0) {
140. printf("未找到姓名为 %s 的学生。\n", name);
141. }
142. else {
143. printf("共找到 %d 条记录。\n", found);
144. }
145. }
146. void queryMenu(Node\* head) {
147. int choice;
148. do {
149. printf("\n--- 查询记录模块 ---\n");
150. printf("  1. 按学号查询\n");
151. printf("  2. 按姓名查询\n");
152. printf("  0. 返回主菜单\n");
153. printf("---------------------\n");
154. printf("请输入您的选择: ");
155. scanf("%d", &choice);
156. clearInputBuffer();
157. switch (choice) {
158. case 1: searchStudentById(head); break;
159. case 2: searchStudentByName(head); break;
160. case 0: break;
161. default: printf("无效选择，请重新输入。\n");
162. }
163. } while (choice != 0);
164. }
165. *// 3. 更新记录模块*
166. void modifyStudent(Node\* head) {
167. if (head->next == NULL) {
168. printf("当前无任何学生信息。\n");
169. return;
170. }
171. char id[20];
172. printf("请输入要修改的学生的学号: ");
173. scanf("%s", id);
174. clearInputBuffer();
175. Node\* p = head->next;
176. int found = 0;
177. while (p != NULL) {
178. if (strcmp(p->data.id, id) == 0) {
179. printf("已找到该学生，请输入新信息:\n");
180. printf("请输入新姓名: ");
181. scanf("%s", p->data.name);
182. clearInputBuffer();
183. printf("请输入新性别: ");
184. scanf("%s", p->data.gender);
185. clearInputBuffer();
186. printf("请输入新年龄: ");
187. scanf("%d", &p->data.age);
188. clearInputBuffer();
189. printf("请输入新班级: ");
190. scanf("%s", p->data.className);
191. clearInputBuffer();
192. printf("\n学生信息修改成功！\n");
193. found = 1;
194. break;
195. }
196. p = p->next;
197. }
198. if (!found) {
199. printf("未找到学号为 %s 的学生。\n", id);
200. }
201. }
202. void deleteStudent(Node\* head) {
203. if (head->next == NULL) {
204. printf("当前无任何学生信息。\n");
205. return;
206. }
207. char id[20];
208. printf("请输入要删除的学生的学号: ");
209. scanf("%s", id);
210. clearInputBuffer();
211. Node\* prev = head;
212. Node\* curr = head->next;
213. int found = 0;
214. while (curr != NULL) {
215. if (strcmp(curr->data.id, id) == 0) {
216. prev->next = curr->next;
217. free(curr);
218. printf("学号为 %s 的学生信息已删除。\n", id);
219. found = 1;
220. break;
221. }
222. prev = curr;
223. curr = curr->next;
224. }
225. if (!found) {
226. printf("未找到学号为 %s 的学生。\n", id);
227. }
228. }
229. void updateMenu(Node\* head) {
230. int choice;
231. do {
232. printf("\n--- 更新记录模块 ---\n");
233. printf("  1. 修改学生信息\n");
234. printf("  2. 删除学生信息\n");
235. printf("  0. 返回主菜单\n");
236. printf("---------------------\n");
237. printf("请输入您的选择: ");
238. scanf("%d", &choice);
239. clearInputBuffer();
240. switch (choice) {
241. case 1: modifyStudent(head); break;
242. case 2: deleteStudent(head); break;
243. case 0: break;
244. default: printf("无效选择，请重新输入。\n");
245. }
246. } while (choice != 0);
247. }
248. *// 4. 统计记录模块*
249. void statisticsByClass(Node\* head) {
250. if (head->next == NULL) {
251. printf("当前无任何学生信息。\n");
252. return;
253. }
254. char className[50];
255. printf("请输入要统计的班级名称: ");
256. scanf("%s", className);
257. clearInputBuffer();
258. Node\* p = head->next;
259. int count = 0;
260. while (p != NULL) {
261. if (strcmp(p->data.className, className) == 0) {
262. count++;
263. }
264. p = p->next;
265. }
266. printf("班级 %s 共有 %d 名学生。\n", className, count);
267. }
268. void statisticsByGender(Node\* head) {
269. if (head->next == NULL) {
270. printf("当前无任何学生信息。\n");
271. return;
272. }
273. int male\_count = 0;
274. int female\_count = 0;
275. Node\* p = head->next;
276. while (p != NULL) {
277. if (strcmp(p->data.gender, "男") == 0) {
278. male\_count++;
279. }
280. else if (strcmp(p->data.gender, "女") == 0) {
281. female\_count++;
282. }
283. p = p->next;
284. }
285. printf("学生总数中，男生有 %d 人，女生有 %d 人。\n", male\_count, female\_count);
286. }
287. void statisticsMenu(Node\* head) {
288. int choice;
289. do {
290. printf("\n--- 统计记录模块 ---\n");
291. printf("  1. 按班级统计\n");
292. printf("  2. 按性别统计\n");
293. printf("  0. 返回主菜单\n");
294. printf("---------------------\n");
295. printf("请输入您的选择: ");
296. scanf("%d", &choice);
297. clearInputBuffer();
298. switch (choice) {
299. case 1: statisticsByClass(head); break;
300. case 2: statisticsByGender(head); break;
301. case 0: break;
302. default: printf("无效选择，请重新输入。\n");
303. }
304. } while (choice != 0);
305. }
306. *// 5. 输出记录模块*
307. void displayAllStudents(Node\* head) {
308. if (head->next == NULL) {
309. printf("当前无任何学生信息。\n");
310. return;
311. }
312. printf("\n======================= 所有学生信息 =======================\n");
313. printf("%-15s %-15s %-10s %-10s %-15s\n", "学号", "姓名", "性别", "年龄", "班级");
314. printf("------------------------------------------------------------\n");
315. Node\* p = head->next;
316. while (p != NULL) {
317. printf("%-15s %-15s %-10s %-10d %-15s\n",
318. p->data.id, p->data.name, p->data.gender, p->data.age, p->data.className);
319. p = p->next;
320. }
321. printf("============================================================\n");
322. }
323. *// 6. 文件操作模块*
324. void saveToFile(Node\* head, const char\* filename) {
325. FILE\* fp = fopen(filename, "w");
326. if (fp == NULL) {
327. printf("打开文件 %s 失败！\n", filename);
328. return;
329. }
330. Node\* p = head->next;
331. while (p != NULL) {
332. fprintf(fp, "%s %s %s %d %s\n",
333. p->data.id, p->data.name, p->data.gender, p->data.age, p->data.className);
334. p = p->next;
335. }
336. fclose(fp);
337. printf("数据已成功保存到 %s 文件中。\n", filename);
338. }
339. Node\* loadFromFile(const char\* filename) {
340. *// 创建一个带头节点的空链表*
341. Node\* head = (Node\*)malloc(sizeof(Node));
342. if (head == NULL) {
343. printf("内存分配失败！\n");
344. exit(1);
345. }
346. head->next = NULL;
347. FILE\* fp = fopen(filename, "r");
348. if (fp == NULL) {
349. printf("未找到数据文件 %s，将创建一个新系统。\n", filename);
350. return head;
351. }
352. Student temp;
353. while (fscanf(fp, "%s %s %s %d %s", temp.id, temp.name, temp.gender, &temp.age, temp.className) != EOF) {
354. Node\* newNode = (Node\*)malloc(sizeof(Node));
355. if (newNode == NULL) {
356. printf("内存分配失败！\n");
357. continue;
358. }
359. newNode->data = temp;
360. newNode->next = NULL;
361. *// 插入到链表尾部*
362. Node\* p = head;
363. while (p->next != NULL) {
364. p = p->next;
365. }
366. p->next = newNode;
367. }
368. fclose(fp);
369. printf("已从 %s 文件加载数据。\n", filename);
370. return head;
371. }
372. *// 辅助函数：释放链表*
373. void freeList(Node\* head) {
374. Node\* p = head;
375. Node\* temp;
376. while (p != NULL) {
377. temp = p;
378. p = p->next;
379. free(temp);
380. }
381. }
382. main.cpp (主函数文件)
383. #define \_CRT\_SECURE\_NO\_WARNINGS
384. #include "student.h"
385. #define FILENAME "student.txt"
386. void showMenu() {
387. printf("\n========= 学生信息管理系统 =========\n");
388. printf("||                                ||\n");
389. printf("||    1. 录入学生信息             ||\n");
390. printf("||    2. 查询学生信息             ||\n");
391. printf("||    3. 更新学生信息             ||\n");
392. printf("||    4. 统计学生信息             ||\n");
393. printf("||    5. 显示所有信息             ||\n");
394. printf("||                                ||\n");
395. printf("||    0. 退出系统 (Exit)          ||\n");
396. printf("||                                ||\n");
397. printf("====================================\n");
398. printf("请输入您的选择 [0-5]: ");
399. }
400. int main() {
401. *// 程序启动时，从文件加载数据*
402. Node\* head = loadFromFile(FILENAME);
403. int choice;
404. do {
405. showMenu();
406. scanf("%d", &choice);
407. clearInputBuffer(); *// 吸收多余的换行符*
408. switch (choice) {
409. case 1:
410. addStudent(head);
411. saveToFile(head, FILENAME);
412. break;
413. case 2:
414. queryMenu(head);
415. break;
416. case 3:
417. updateMenu(head);
418. saveToFile(head, FILENAME);
419. break;
420. case 4:
421. statisticsMenu(head);
422. break;
423. case 5:
424. displayAllStudents(head);
425. break;
426. case 0:
427. {   FILE\* fp = fopen(FILENAME, "w"); }*// 以写入模式打开文件，会直接清空内容*
428. printf("正在退出系统...感谢使用！\n");
429. break;
430. default:
431. printf("无效的输入，请输入 0-5 之间的数字。\n");
432. break;
433. }
434. printf("\n按 Enter 键继续...");
435. getchar(); *// 暂停程序，等待用户按键*
436. system("cls"); *// 清屏*
437. } while (choice != 0);
438. *// 退出程序前，释放链表内存*
439. freeList(head);
440. return 0;
441. }