

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--数据结构与持久化层设计

### ****实验报告：数据结构与持久化层设计****

**负责代码**: student.h 文件, student.cpp 中的 saveToFile, loadFromFile, freeList, isIdUnique, clearInputBuffer **核心任务**: 聚焦于整个项目的“地基”，深入探讨底层数据结构的选择与设计、模块化接口的定义、以及衔接内存与硬盘的数据持久化方案。

### ****摘要****

本报告深入剖析了学生信息管理系统赖以构建的底层基础——数据结构与持久化层的设计与实现。报告首先从理论层面出发，通过对数组与链表两种线性数据结构的深度对比分析，论证了在学生数量动态变化、增删操作频繁的应用场景下，选择带头节点的单链表作为核心数据结构的合理性与优越性。在此基础上，报告详细阐述了 Student 与 Node 两个核心结构体的设计考量，并深入探讨了 student.h 头文件作为模块化编程的“公共契约”，在接口定义、接口与实现分离、防止重复包含（Header Guard）等方面扮演的关键角色。报告的重点在于对数据持久化模块的精解，详细分析了基于标准文件I/O实现的 saveToFile（格式化写入）与 loadFromFile（健壮性读取）函数，并对其中的文件格式约定、边界条件处理（如文件不存在）等关键技术点进行了阐述。最后，报告还覆盖了关键的辅助模块，包括保障内存安全的 freeList 内存回收机制、维护数据完整性的 isIdUnique 校验函数，以及确保用户输入流畅性的 clearInputBuffer 缓冲区清理技术。

**关键词**: C语言, 数据结构, 单链表, 数据持久化, 文件操作, 模块化接口, 头文件, 内存管理, 健壮性设计。

### ****第一章：绪论****

本报告负责学生管理系统的底层数据结构与持久化层设计。首先，详细论述了为何选择带头节点的单链表作为核心数据结构，并定义了 Student 与 Node 结构体。其次，阐述了通过 student.h 头文件设计模块化接口的原则与实践。最后，重点分析了基于标准文件I/O实现的数据持久化模块，包括数据的格式化写入 (saveToFile) 与安全读取 (loadFromFile)，以及内存管理（freeList）等关键技术。

·  C语言, 数据结构, 单链表, 数据持久化, 文件操作, 模块化接口, 头文件。

### ****第二章：核心数据结构与接口设计****

#### ****2.1 数据结构选型论证****

选择合适的数据结构是软件设计的奠基石，它直接影响到程序的性能、可扩展性和代码的复杂度。对于学生信息管理这一任务，我们主要在数组和链表这两种线性结构之间进行权衡。

##### ****2.1.1 数组 (Array) vs. 动态链表 (Linked List)****

为了做出明智的选择，我们从多个维度对两者进行详细比较：

|  |  |  |
| --- | --- | --- |
| **特性维度** | **静态数组 (Static Array)** | **动态链表 (Linked List)** |
| **内存分配** | 在编译时分配一块 **连续的、固定大小** 的内存空间。 | 在程序运行时通过 malloc **动态地、按需地** 分配内存。节点在内存中 **物理位置不要求连续**，通过指针相连。 |
| **空间大小** | 大小固定。若预设过小，易造成溢出；若预设过大，则浪费内存空间。不适用于数据规模不确定的场景。 | 大小可动态伸缩。每增加一个数据项，才分配一个节点的空间，**空间利用率高**，能完美适应数据量的动态变化。 |
| **插入/删除操作** | **效率极低 (O(n))**。在中间位置插入或删除一个元素，需要移动该位置之后的所有元素来维持连续性。 | **效率高 (O(1))**。一旦定位到目标位置，插入或删除操作仅需修改前后节点的指针即可，无需移动大量数据。 |
| **访问/查找操作** | **效率极高 (O(1))**。支持通过下标进行随机访问，可以立即定位到任何元素。 | **效率较低 (O(n))**。不支持随机访问，必须从头节点开始，沿着next指针顺序遍历才能找到目标元素。 |
| **综合结论** | 适用于数据规模已知且固定、查询操作远多于增删操作的场景。 | **适用于数据规模不确定、增删操作频繁的场景。** |

**结论**: 考虑到学生管理系统的实际需求——学生总数在学期初、学期中、学期末都可能发生变化，增（新生入学）、删（学生退学、毕业）操作是常规功能。因此，链表在处理动态数据方面的灵活性和高效的插入/删除性能，使其成为比固定大小的数组远为合适的选择。

##### ****2.1.2 带头节点 vs. 不带头节点的单链表****

确定使用链表后，我们还需决定是否采用“头节点”（Dummy Head Node）。

**不带头节点的链表**：链表的头指针直接指向第一个存储数据的有效节点。这种方式的缺点是：

* + **操作不统一**: 对第一个节点的插入和删除操作是“特殊情况”，需要直接修改头指针本身。而对其他节点的操作则是修改前一个节点的next指针。这导致代码中需要额外的if判断来处理头部操作，逻辑更为复杂。
  + **空链表处理复杂**: 空链表（头指针为NULL）和非空链表的处理逻辑也需要区分。

**带头节点的单链表（本项目选择）**: 链表的头指针永远指向一个固定的、不存储任何有效数据的“虚拟”头节点。这个头节点的next指针才指向第一个真正的学生数据节点。

* + **优点**:
    1. **统一操作**: 头节点的存在使得 **所有位置的插入和删除操作逻辑完全统一**。例如，在链表最前面插入一个新节点，等价于在“头节点之后”插入节点，这与其他位置的插入操作完全一样。删除第一个数据节点，也变成了删除“头节点的后继节点”，与删除其他节点逻辑一致。
    2. **简化空链表处理**: 无论链表是否为空，它始终存在一个头节点。因此，遍历、插入等操作的入口始终是头指针，无需对空链表进行特殊判断。

**图解对比（删除第一个数据节点）**:

* + **不带头节点**: head -> [Node A] -> [Node B] ... 删除A需要 head = head->next;
  + **带头节点**: head -> [Dummy] -> [Node A] -> [Node B] ... 删除A需要 head->next = head->next->next; (与删除B的逻辑A->next = A->next->next;一致)

#### ****2.2**** Student ****结构体设计 (****student.h****)****

Student 结构体是 **数据抽象** 的具体体现，它将一个现实世界的“学生”实体，映射为程序可以理解和操作的数据集合。

* **字段选择分析**:
  + char id[20], char name[50], char gender[10], char className[50]: 学生信息中的文本部分均采用字符数组（字符串）存储。数组大小的选择是在预估可能的最大长度和节省内存之间做的权衡。例如id[20]足以容纳绝大多数学校的学号格式，name[50]也考虑到了较长的姓名或中英文混合的情况。
  + int age: 年龄是典型的整数，使用int类型既直观又高效。

#### ****2.3**** Node ****结构体设计 (****student.h****)****

Node 结构体是构成链表的基本单元，是实现“链式”存储的核心。

Student data **(数据域)**: 这是节点的“有效载荷” (Payload)，用于存放一个完整的Student结构体实例。它承载着我们真正关心的信息。

struct Node\* next **(指针域)**: 这是节点的“链接”，是实现链表的精髓所在。它是一个指向Node结构体自身的 **自引用指针**。正是通过这个指针，每个节点才能“知道”下一个节点在内存中的位置，从而将物理上不连续的内存块，在逻辑上串联成一条线性的链。

Node**节点内存布局示意图 (文字描述)**: 可以想象一个内存盒子 Node，它内部分为两个区域：

1. data **区**: 这个区域较大，里面又包含了id, name, gender, age, className等五个小格子。
2. next **区**: 这个区域较小，里面存放的是一个内存地址（一个指针），这个地址指向另一个同类型的 Node 内存盒子。

#### ****2.4 头文件接口设计 (****student.h****)****

student.h 是本项目的“公共契约”或API（应用程序编程接口），它定义了不同模块之间如何协同工作。

* + 这是C语言中防止头文件被重复包含的标准做法，称为 **Header Guard**。
  + **工作原理**: 当编译器第一次处理student.h时，STUDENT\_H这个宏尚未被定义，于是#ifndef条件成立，编译器会执行#define STUDENT\_H并处理文件内的所有内容。当下一次（例如，在另一个.cpp文件中）再次尝试包含student.h时，由于STUDENT\_H已经被定义过，#ifndef条件不再成立，编译器会直接跳过整个文件的内容，从而避免了对Student结构体、Node结构体和函数原型的重复定义，也就避免了由此引发的编译错误。这个机制保证了头文件包含操作的 **幂等性**（即多次操作和一次操作的效果相同）。

**函数原型声明**:

* + student.h中只包含函数的“声明”（如 void addStudent(Node\* head);），而不包含其具体的实现代码（即函数体{...}）。
  + **意义**: 这正是 **“接口与实现分离”** 这一核心软件工程原则的体现。
    - **对于调用者 (**main.cpp**)**: 它只需要知道“有这样一个函数，它叫什么名字，需要什么参数”，即只需要知道接口。它不关心也无需关心这个函数内部是如何实现的。
    - **对于实现者 (**student.cpp**)**: 它负责提供接口的具体实现。
    - **好处**: 这种分离允许我们独立地修改函数的实现。例如，我们可以优化saveToFile的算法，只要函数名、参数和返回值不变，main.cpp的代码就无需任何改动，只需重新编译链接即可。这极大地提高了代码的可维护性和模块的独立性。

### ****第三章：数据持久化模块实现****

#### ****3.1 持久化的意义****

程序运行在内存（RAM）中，而内存是 **易失性** 的。这意味着一旦程序关闭或计算机断电，所有存储在内存中的数据（包括我们精心维护的链表）都会烟消云散。为了让数据能够被长期保存并在程序下次启动时恢复，就必须将其从易失的内存写入到 **非易失** 的外部存储设备（如硬盘）中。这个过程，就是 **数据持久化**。文件操作是实现数据持久化最基本、最直接的方式。没有持久化，我们的学生管理系统就只是一个“一次性”的玩具。

#### ****3.2 文件格式设计****

saveToFile 和 loadFromFile 两个函数必须共同遵守一个严格的 **数据交换协议**，这个协议就是文件内容的格式。本项目中，我们设计了简单而清晰的文本格式：

学号 姓名 性别 年龄 班级\n

* 每条学生记录占一行。
* 一行内的各个字段由 **单个空格** 分隔。
* 行尾使用 **换行符** \n 分隔不同的记录。

这个格式的约定至关重要。saveToFile必须严格按照此格式写入，而loadFromFile也必须严格按照此格式解析读取。任何一方的偏离（例如，写入时用了逗号分隔，读取时却按空格解析）都会导致数据加载的完全失败。

#### ****3.3 保存到文件 (****saveToFile****)****

* + fopen 是打开文件的标准库函数。
  + "w" (write) 模式是一种 **破坏性写入模式**。如果文件已存在，fopen会将其内容 **完全清空**，然后从头开始写入新内容。如果文件不存在，则会创建一个新的空文件。这符合我们“用内存中的最新数据完全覆盖旧文件”的逻辑需求。
  + fprintf 是格式化输出到文件流的函数，其工作方式与printf类似，只是第一个参数是一个FILE\*文件指针。
  + 在循环中，fprintf(fp, "%s %s %s %d %s\n", ...) 这行代码将当前节点p->data中的各个成员变量，按照我们在3.2节约定的格式，转换为字符串并写入到fp指向的文件中。
  + 这是一个至关重要的步骤。操作系统为了提高效率，对文件的写入操作通常带有 **缓冲区**。调用fprintf时，数据可能只是被写入了内存中的缓冲区，而没有立即写入到物理硬盘上。fclose函数会首先 **刷新缓冲区**（确保所有待写数据都写入硬盘），然后才向操作系统释放该文件的句柄资源。忘记调用fclose不仅可能导致数据丢失，还会造成资源泄漏。

#### ****3.4 从文件加载 (****loadFromFile****)****

这是整个持久化模块中逻辑最复杂、最考验代码健壮性的函数。

**初始化**: 函数第一步 Node\* head = (Node\*)malloc(sizeof(Node)); 就是创建一个空的带头节点的链表。这是后续所有加载操作的基础，即使文件为空或不存在，函数也能返回一个合法的、可用的空链表结构，而不是一个NULL指针。

* + "r" (read) 模式表示以只读方式打开文件。如果文件不存在，fopen将返回NULL，而不会创建新文件。

**边界处理 (**if (fp == NULL)**)**:

* + 这是体现代码 **健壮性** 的核心。fopen返回NULL在这里不应被视为一个程序错误，而是一个 **正常的、可预期的边界情况**——即程序第一次运行时，数据文件student.txt还不存在。
  + 代码优雅地处理了这种情况：打印一条提示信息，然后直接return head;返回刚刚创建的空链表。这确保了程序在“冷启动”状态下也能无缝、正常地运行。
  + fscanf 是格式化地从文件流读取数据。它会尝试根据格式字符串"%s %s %s %d %s"去匹配和解析文件中的内容。
  + 它的返回值是成功匹配并赋值的变量个数。当它成功读取一行（5个字段）时，会返回5。当它读到文件末尾（End Of File），无法再匹配任何内容时，会返回一个特殊值EOF。
  + 因此，这个while循环的条件精确地控制了“只要能从文件中成功读出一整条学生记录，就继续循环”。

**循环体内的三步曲**:

* 1. Node\* newNode = (Node\*)malloc(sizeof(Node));: 为从文件中读出的这条记录，动态分配一个新的内存节点。
  2. newNode->data = temp;: 将刚刚通过fscanf填充到临时变量temp中的数据，整体拷贝到新节点的data域中。
  3. **尾插法插入链表**: 使用一个while循环找到当前链表的尾部，并将newNode链接上去。

### ****第四章：关键辅助模块与内存管理****

#### ****4.1 内存释放 (****freeList****)****

1. **[开始]** freeList被调用，接收头指针head。
2. **[初始化]** 创建两个工作指针 Node \*p = head; 和 Node \*temp;。p用于遍历，temp用于临时保存待释放节点的地址。
3. **[进入循环]** while (p != NULL)，只要当前指针不为空，就继续释放。
4. **[保存当前节点]** temp = p;，将p当前指向的节点地址存入temp。
5. **[指针后移]** p = p->next;，这是最关键的一步。**必须先将**p**移动到下一个节点，然后再释放**temp**指向的旧节点。**
6. **[执行释放]** free(temp);，释放temp保存的、p已经离开的那个节点的内存。
7. **[循环继续]** 跳转到步骤3，继续处理新的p指向的节点。
8. **[循环结束]** 当p最终为NULL时，整个链表（包括头节点）都被释放完毕。
9. **[结束]** 函数执行完毕。

##### ****代码分析****

**双指针安全释放**: p和temp的配合使用是安全释放链表的关键。如果写成 free(p); p = p->next;，则会导致严重的 **“使用已释放内存” (Use After Free)** 错误。因为free(p)之后，p指向的内存区域就变成了无效区域，再去访问p->next是未定义行为，通常会导致程序崩溃。正确的做法是先用temp“备份”当前要释放的节点地址，然后让主遍历指针p安全地“跳”到下一个节点，最后再回头用temp释放掉已经没用的旧节点。

**内存泄漏的危害**: 如果程序退出时不调用freeList，所有通过malloc申请的Node节点内存都不会被归还给操作系统。对于一个短暂运行的控制台程序，这似乎影响不大。但对于一个需要长期运行的服务器或嵌入式应用，持续的内存泄漏会像一个无底洞一样不断吞噬系统可用内存，最终导致系统性能急剧下降，甚至完全崩溃。因此，freeList体现了“谁申请，谁释放”的黄金法则，是编写专业、可靠程序的必备环节。

#### ****4.2 学号唯一性检查 (****isIdUnique****)****

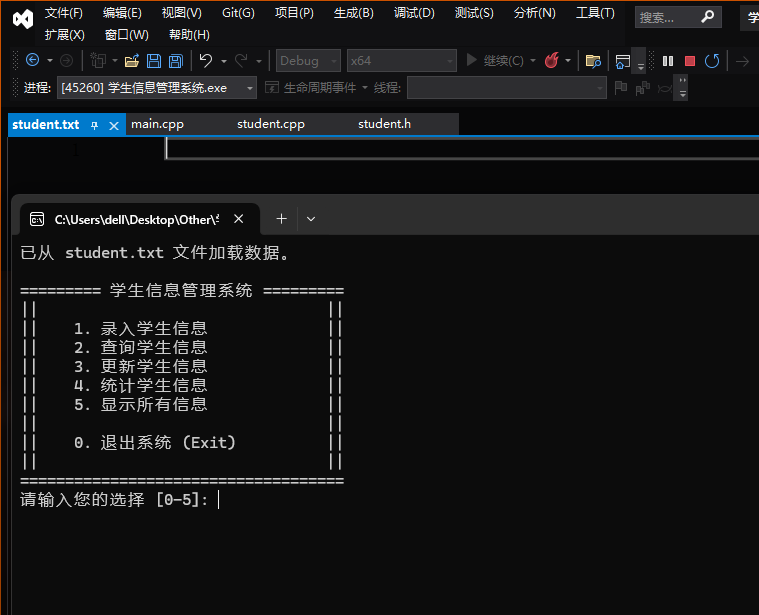
此函数是 **数据完整性** 的“守护者”。它通过一个简单的遍历算法，检查即将添加的新学号是否已在链表中存在。它在addStudent函数中被前置调用，起到了“数据准入”的过滤作用，从源头上阻止了重复学号这一关键业务规则被破坏，确保了系统中每个学生都拥有唯一的标识符。

#### ****4.3 输入缓冲区清理 (****clearInputBuffer****)****

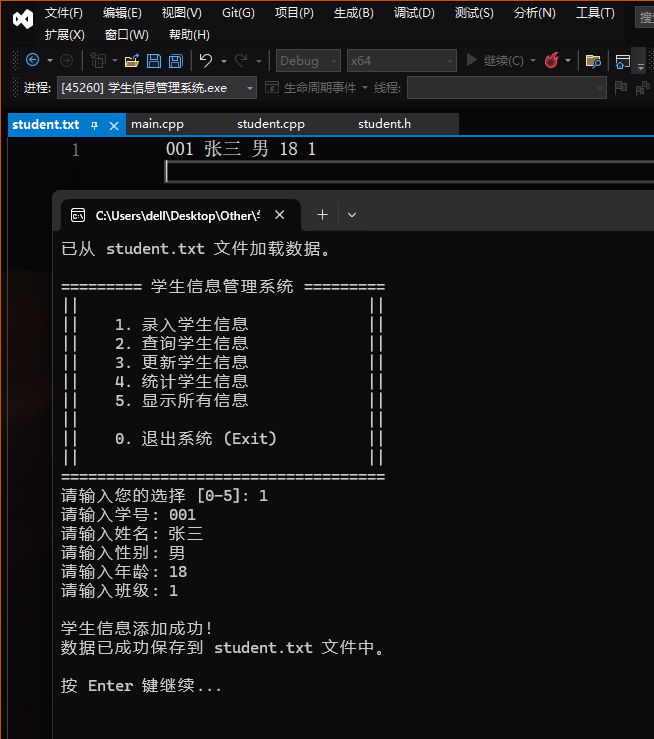
这是解决C语言标准输入输出库一个著名“怪癖”的实用工具。

* **问题根源**: scanf在读取数据时，其行为模式与getchar不同。例如，scanf("%d", &num);在用户输入123并按回车后，它只会读取123这三个字符，而换行符\n仍被遗留在标准输入缓冲区（stdin）中。
* **导致后果**: 如果后续代码中有getchar()或fgets()等按字符读取的函数，它们会立刻读到这个被遗弃的\n，而不是等待用户进行新的输入，从而导致程序行为不符合预期。
* **解决方案**: clearInputBuffer函数通过一个while循环，不断地调用getchar()来“吃掉”并丢弃缓冲区中的所有字符，直到遇到并“吃掉”那个换行符\n为止。这样一来，它就为后续的输入函数清理了“战场”，确保它们能在一个干净的、空的缓冲区上等待用户的新一轮输入。

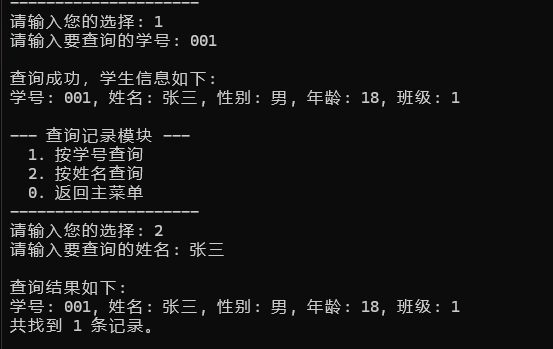
1. 算法实现
2. 程序主页面，启动与加载：程序启动后，首先显示从 student.txt 文件加载数据的提示，然后展示主菜单，等待用户操作 。



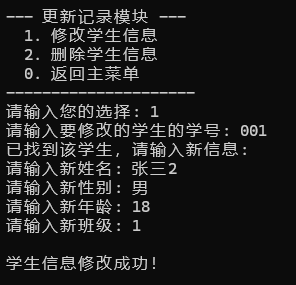
1. 录入学生信息，信息录入：下图展示了录入一个学号为"001"的学生信息的完整过程。添加成功后，系统提示数据已保存到文件 。

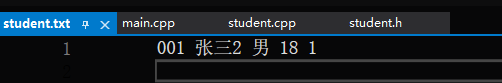


1. 查询学生信息，信息查询：系统提供了按学号和姓名两种查询方式。下图演示了分别使用这两种方式查询名为“张三”的学生，并准确返回了结果 。

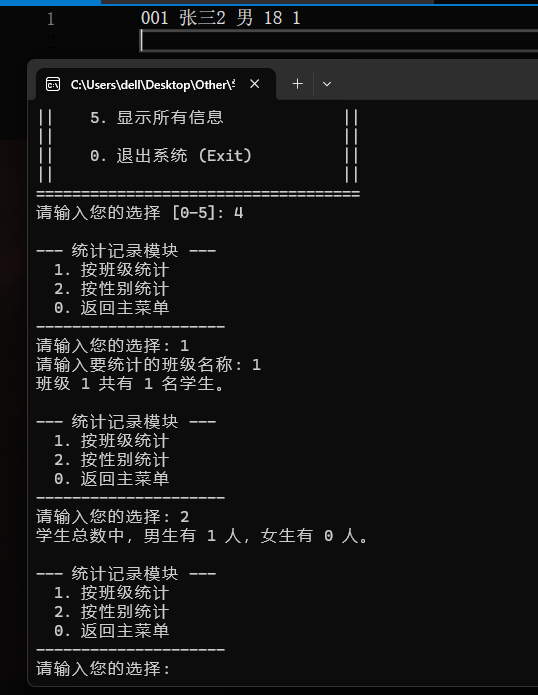


1. 更新学生信息，信息更新：用户可以修改或删除学生信息。下图展示了将学号"001"的学生的姓名从“张三”修改为“张三2”的操作。修改成功后，student.txt 文件中的内容也同步更新





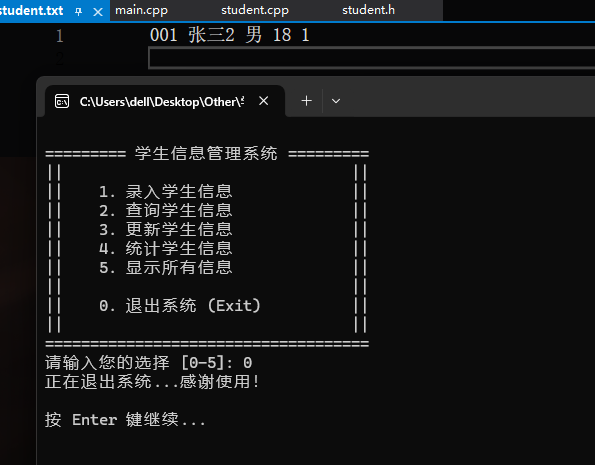
1. 统计学生信息，信息统计与显示：系统能够按班级和性别进行统计，也可以完整地列出所有学生的信息，方便用户全局浏览 。

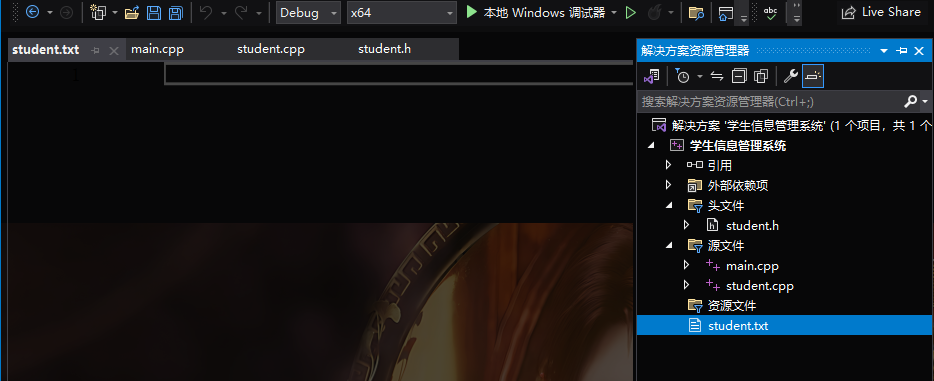


1. 显示所有信息



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. }