项目说明文档

数据结构课程设计

——考试报名系统

作 者 姓 名：张翔

学　　号：2352985

指 导 教 师：张颖

学院、 专业：计算机科学与技术学院 软件工程

同济大学

Tongji University

二〇二四 年 十二 月 七 日

1 项目分析

1.1 项目背景分析

考试报名是高校教务管理中一项至关重要的工作。随着教育系统的不断发展和进步，考试报名系统也需要不断升级和改进以适应现代社会的需求。

考试报名是一项庞大的工作，涉及多个环节和大量考生信息的管理。传统的手工管理已经无法满足快速、高效的需求。考生的信息包括考号、姓名、性别、年龄、报考类别等多种属性。这些信息需要被准确、高效地录入、修改、查询和删除。

1.2 项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)实现对考生信息的录入、输出、查询、添加、修改和删除等功能，确保数据的准确、高效管理；

(2)设计简单直观的控制台界面，使操作便捷、容易上手，适应不同用户的操作习惯；

(3)选择合适的数据结构，以支持对考生信息的高效操作，同时考虑信息的关联性和复杂度；

(4)实现异常处理机制，确保系统稳定性和安全性，避免因用户输入错误导致系统崩溃或信息丢失；

(5)设计系统以支持未来的扩展和功能增加，满足不同用户、不同应用场景下的需求。

1.3 项目功能分析

本项目旨在通过模拟考试报名管理过程，实现对考生信息的录入、输出、查询、添加、修改和删除等功能，从而实现对考生信息的高效管理。需要设计合适的数据结构、开发用户友好的控制台界面，并考虑系统的稳定性、安全性以及未来的扩展性。通过该项目的实施，可以提高考试报名管理的效率和准确性，为教务管理部门和考生提供更好的服务。下面对项目的功能进行详细分析。

1.3.1 录入考生信息功能

允许用户输入考生的基本信息，包括考号、姓名、性别、年龄、报考类别等，并建立考生信息系统。程序需要验证输入的信息是否符合规范，例如考号是否为由若干数字字符组成的字符串、年龄是否为正整数等。

1.3.2 输出考生信息功能

能够输出已录入的考生信息，包括考号、姓名、性别、年龄、报考类别等。

1.3.3 插入考生功能

允许用户在已有考生信息的基础上继续添加新的考生信息，包括考号、姓名、性别、年龄、报考类别等。

1.3.4 删除考生功能

允许用户根据考号等关键信息选择要删除的考生信息，进行考生信息的删除操作。

1.3.5 查询考生功能

允许用户通过考号等关键信息进行查询，程序能够返回符合条件的考生信息。

1.3.6 修改考生功能

允许用户根据考号等关键信息选择要修改的考生信息，可以修改考生的姓名、性别、年龄、报考类别等。

1.3.7 统计考生功能

允许用户在考生信息系统中统计考生信息，以对所有考生的信息有更全面的掌握和了解。

1.3.8 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

2 项目设计

2.1 数据结构设计

基于项目分析，考试报名系统的设计中选择使用链表作为数据结构而不是数组，主要基于以下几个考虑：

⑴动态大小需求：链表可以动态地分配内存，适应不同数量的考生信息，而数组需要预先确定大小，可能会导致内存浪费或不足；

⑵插入和删除操作效率高：链表对于插入和删除操作效率较高，因为只需要调整节点的指针即可，而数组需要移动元素，时间复杂度效率较高；

⑶频繁的数据修改：如果考生信息需要频繁修改，例如修改报名信息、取消报名等，链表更适合，因为修改节点的指针比修改数组元素更高效；

⑷不需要随机访问：如果系统不需要通过索引随机访问考生信息，而只是按顺序处理，链表可以满足需求；

链表适合在需要动态调整大小、频繁插入和删除操作以及不需要随机访问的情况下使用，而数组更适合需要随机访问和固定大小的情况。基于上述分析，在设计考试报名系统时，选择链表作为数据结构更合适。

2.2 结构体与类设计

2.2.1 MyLinkNode 结构体的设计

2.2.1.1 概述

MyLinkNode 是一个模板结构体，用于实现单链表的节点，每个节点存储一个类型为 Type 的数据，并包含一个指向下一个节点的指针。它提供了两个构造函数，一个是默认构造函数用于初始化 next 指针为 nullptr，另一个构造函数则可以同时初始化节点的数据和 next 指针。

2.2.1.2 结构体定义

```
template <typename Type>
struct MyLinkNode
{
    Type data;
    MyLinkNode<Type>* next;
    MyLinkNode(MyLinkNode<Type>* ptr = nullptr) { next = ptr; }
    MyLinkNode(const Type& item, MyLinkNode<Type>* ptr = nullptr) { data = item; next = ptr; }
};
```

2.2.2 MyList 类的设计

2.2.2.1 概述

MyList 是一个模板类，基于单链表实现，提供了链表的基本操作。它通过 head 和 tail 指针维护链表的起始和末尾，并提供了多个成员函数来操作链表，如获取链表长度、获取头尾节点、查找和定位元素、插入和删除节点、以及获取和设置指定位置的数据。此外，MyList 还包含判断链表是否为空的功能。使用户能够方便地管理和操作链表数据。

2.2.2.2 类定义

```
template <typename Type>
class MyList
{
private:
```

```
    MyLinkNode<Type>* head;
    MyLinkNode<Type>* tail;
public:
    MyList();
    ~MyList();
    int getLength(void) const;
    MyLinkNode<Type>* getHead(void) const;
    MyLinkNode<Type>* getTail(void) const;
    MyLinkNode<Type>* search(Type item) const;
    MyLinkNode<Type>* locate(int i) const;
    bool getData(int i, Type& item) const;
    bool setData(int i, Type& item);
    bool insert(int i, Type& item);
    bool remove(int i, Type& item);
    bool isEmpty(void) const;
};
```

2.2.3 Student 结构体的设计

2.2.3.1 概述

Student 是一个用于存储考生信息的结构体，包含五个成员变量：examID 用于存储考生的考试编号，长度为 MAX_ID_LENGTH + 1；name 用于存储考生的姓名，长度为 MAX_NAME_LENGTH + 1；examType 用于存储考生的考试类型，长度为 MAX_TYPE_LENGTH + 1；gender 用于表示考生的性别，true 代表男性，false 代表女性；age 表示考生的年龄，类型为 int。

2.2.3.2 结构体定义

```
struct Student
{
    char examID[MAX_ID_LENGTH + 1] = { 0 };
    char name[MAX_NAME_LENGTH + 1] = { 0 };
    char examType[MAX_TYPE_LENGTH + 1] = { 0 };
    bool gender = true;
    int age = 0;
};
```

2.2.4 Manager 类的设计

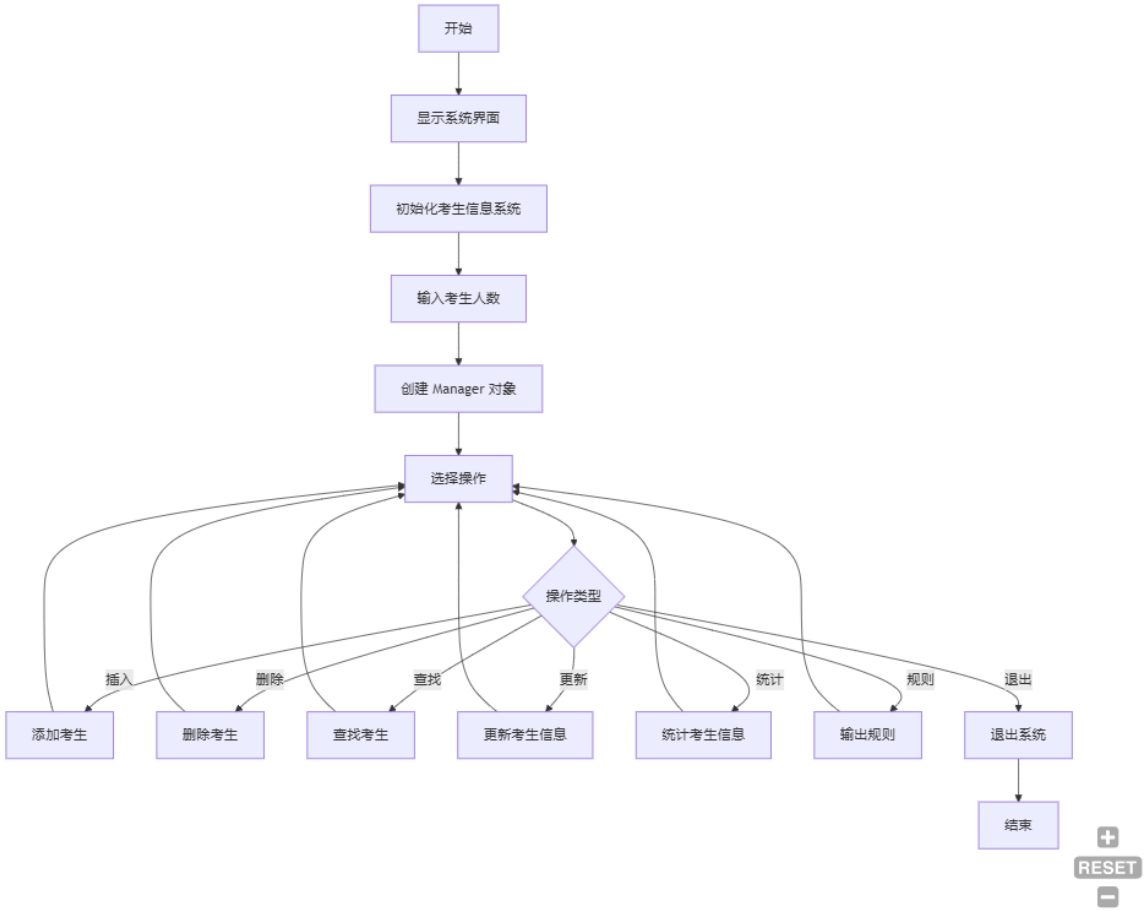
2.2.4.1 概述

Manager 类是一个用于管理考生信息的类,包含一个 MyList<Student>类型的成员变量 student，用于存储考生的数据。该类提供了多个功能函数，包括构造函数用于初始化系统，buildStudentList 用于批量添加考生信息，studentInput 用于输入考生详细信息，addStudent、deleteStudent、findStudent 和 updateStudent 分别用于添加、删除、查找和更新考生数据。它还包含 statisticsFunction 用于展示所有考生信息，以及 outputRules 用于输出系统规则。

2.2.4.2 类定义

```
class Manager {
private:
    MyList<Student> student;
public:
    Manager(int stuNum);
    void buildStudentList(int stuNum);
    int findPosByStuNo(const char id[]);
    int GetPosByStuNo(const char* prompt);
    Student studentInput();
    void addStudent();
    void deleteStudent();
    void findStudent();
    void updateStudent();
    void statisticsFunction();
    void outputRules();
};
```

2.3 项目主体架构设计



2.3.1 项目主体架构流程图

3 项目功能实现

3.1 项目主体架构的实现

3.1.1 项目主体架构实现思路

实现了一个简单的考试报名系统，通过一个菜单驱动的方式让用户选择不同的操作，如添加、删除、查找、更新学生信息等。具体实现思路如下：程序首先初始化考生信息系统并获取考生人数，然后进入一个无限循环，用户根据提示选择操作，程序根据选择执行相应的管理功能（如 Manager 类中的 addStudent、deleteStudent 等函数），当用户输入无效操作时，系统会退出。通过 Manager 类封装具体操作，便于扩展和维护。

3.1.2 项目主体架构核心代码

```
int main()
{
    /* 进入考试报名系统 */
    std::cout << "+-----------------------------+" << std::endl;
    std::cout << "|        考生报名系统         |" << std::endl;
    std::cout << "|  Exam Registration System   |" << std::endl;
    std::cout << "+-----------------------------+" << std::endl;
    std::cout << std::endl << ">>> 正在初始化考生信息系统" << std::endl;
    int studentNum = inputInteger(1, INT_MAX, "考生人数");
    Manager manager(studentNum);
    while (true) {
        int operation = int(selectOperation());
        if (operation == Insert)
            manager.addStudent();
        else if (operation == Delete)
            manager.deleteStudent();
        else if (operation == Find)
            manager.findStudent();
        else if (operation == Update)
            manager.updateStudent();
        else if (operation == Count)
            manager.statisticsFunction();
        else if (operation == Rule)
            manager.outputRules();
        else {
            std::cout << std::endl << ">>> 考生报名系统已退出" << std::endl;
            break;
        }
    }
    /* 程序退出 */
```

```
    return 0;
}
```

3.2 录入考生信息功能的实现

3.2.1 录入考生信息功能实现思路

Manager::buildStudentList 函数的实现思路是通过提示用户输入考生信息来初始化考生列表。首先，函数通过 inputPrompt 显示提示信息，接着使用循环根据输入的考生数量（stuNum）逐个获取考生信息，并通过 student.insert 方法将每个考生的 Student 对象插入到学生列表中。初始化完成后，函数输出初始化成功的消息，并使用 printHeader 打印标题。然后，另一个循环遍历所有考生，调用 student.getData 获取每个考生的详细信息，并通过 printStuinfo 函数打印每个考生的详细资料。最后，调用 printFooter 输出结束标志。

3.2.2 录入考生信息功能核心代码

```
void Manager::buildStudentList(int stuNum)
{
    inputPrompt("全部考生");
    for (int count = 1; count <= stuNum; count++) {
        Student temp = studentInput();
        student.insert(count, temp);
    }
    std::cout << std::endl << ">>> 考生系统初始化完成 (考生总数: " << stuNum << " )" << std::endl;
    printHeader("全部考生的详细信息如下");
    for (int i = 1; i <= stuNum; i++) {
        Student temp;
        student.getData(i, temp);
        printStuinfo(temp);
    }
    printFooter();
}
```

3.3 插入考生功能的实现

3.3.1 插入考生功能实现思路

Manager::addStudent 函数的具体实现思路如下：首先，函数通过 inputInteger 提示用户输入插入位置，位置范围是从 1 到当前学生列表长度加 1，以确保插入位置合法。接着，调用 inputPrompt 显示新增考生的提示信息，并通过 studentInput 函数获取用户输入的考生数据，生成一个 Student 对象。最后，使用 student.insert 方法将新考生插入到指定位置。整体流程是通过用户输入获取新考生数据，并将其插入到列表中。

3.3.2 插入考生功能核心代码

```
void Manager::addStudent()
{
    int pos = inputInteger(1, student.getLength() + 1, "插入考生的位置");
    inputPrompt("新增考生");
    Student temp = studentInput();
    student.insert(pos, temp);
}
```

3.4 删除考生功能的实现

3.4.1 删除考生功能实现思路

Manager::deleteStudent 函数的具体实现思路如下：首先，函数检查学生列表是否为空，如果为空，输出提示并返回，避免执行删除操作。接着，调用 GetPosByStuNo 函数根据考生学号获取待删除考生的位置。如果没有找到对应的考生（即返回位置为 0），则输出提示信息并结束操作。若找到对应的考生，则使用 student.remove 函数删除该位置的考生数据，并将被删除的考生信息存储在 temp 中。整体流程是先验证数据存在性，再执行删除操作。

3.4.2 删除考生功能核心代码

```
void Manager::deleteStudent()
{
    if (student.getLength() == 0) {
        std::cout << std::endl << ">>> 考生信息表为空，无法执行删除操作!" << std::endl;
        return;
    }
```

```
    int pos = GetPosByStuNo("需删除的考生");
    if (pos == 0) {
        std::cout << std::endl << ">>> 未查询到该考生" << std::endl;
        return;
    }
    Student temp;
    student.remove(pos, temp);
}
```

3.5 查询考生功能的实现

3.5.1 查询考生功能实现思路

Manager::findStudent 函数的具体实现思路如下：首先，函数检查学生信息表是否为空，如果为空，输出提示并返回，避免进行查询操作。接着，调用 GetPosByStuNo 函数根据考生学号获取待查询考生的位置。如果没有找到对应的考生（即返回位置为 0），则输出提示信息并结束操作。若找到该考生，函数使用 student.getData 获取该位置的考生信息，并将其存储在 temp 中。然后，调用 printHeader 和 printStuinfo 函数打印该考生的详细信息，最后调用 printFooter 输出结束标志。整体流程是先验证数据存在性，再执行查询并展示结果。

3.5.2 查询考生功能核心代码

```
void Manager::findStudent()
{
    if (student.isEmpty()) {
        std::cout << std::endl << ">>> 考生信息表为空!" << std::endl;
        return;
    }
    int pos = GetPosByStuNo("需查询的考生");
    if (pos == 0) {
        std::cout << std::endl << ">>> 未查询到该考生" << std::endl;
        return;
    }
    Student temp;
    student.getData(pos, temp);
```

```
    printHeader("该考生的详细信息如下");
    printStuinfo(temp);
    printFooter();
}
```

3.6 修改考生功能的实现

3.6.1 修改考生功能实现思路

Manager::updateStudent 函数的具体实现思路如下：首先，函数检查学生信息表是否为空，如果为空，输出提示并返回，避免执行修改操作。接着，调用 GetPosByStuNo 函数根据考生学号获取待修改考生的位置。如果没有找到对应的考生（即返回位置为 0），则输出提示信息并结束操作。若找到该考生，函数通过 inputPrompt 提示用户输入新的考生信息，并使用 studentInput 函数获取新的数据，生成一个 Student 对象。最后，调用 student.setData 方法将新数据更新到指定位置的考生信息中。整体流程是验证数据存在性后，获取并更新考生的信息。

3.6.2 修改考生功能核心代码

```
void Manager::updateStudent()
{
    if (student.isEmpty()) {
        std::cout << std::endl << ">>> 考生信息表为空!" << std::endl;
        return;
    }
    int pos = GetPosByStuNo("需修改的考生");
    if (pos == 0) {
        std::cout << std::endl << ">>> 未查询到该考生" << std::endl;
        return;
    }
    inputPrompt("需修改的考生");
    Student temp = studentInput();
    student.setData(pos, temp);
}
```

3.7 统计考生功能的实现

3.7.1 统计考生功能实现思路

Manager::statisticsFunction 函数的具体实现思路如下：首先，函数检查学生信息表是否为空，如果为空，输出提示并返回，避免执行统计操作。接着，调用 printHeader 打印标题，表示接下来将展示所有考生的详细信息。然后，使用循环遍历学生列表，通过 student.getData 获取每个考生的信息，并调用 printStuinfo 函数打印每个考生的详细资料。最后，调用 printFooter 输出结束标志，完成统计输出。整体流程是检查数据存在性后，遍历所有考生并展示其信息。

3.7.2 统计考生功能核心代码

```
void Manager::statisticsFunction()
{
    if (student.isEmpty()) {
        std::cout << std::endl << ">>> 考生信息表为空!" << std::endl;
        return;
    }
    printHeader("全部考生的详细信息如下");
    for (int i = 1; i <= student.getLength(); i++) {
        Student temp;
        student.getData(i, temp);
        printStuinfo(temp);
    }
    printFooter();
    std::cout << std::endl;
}
```

4 项目测试

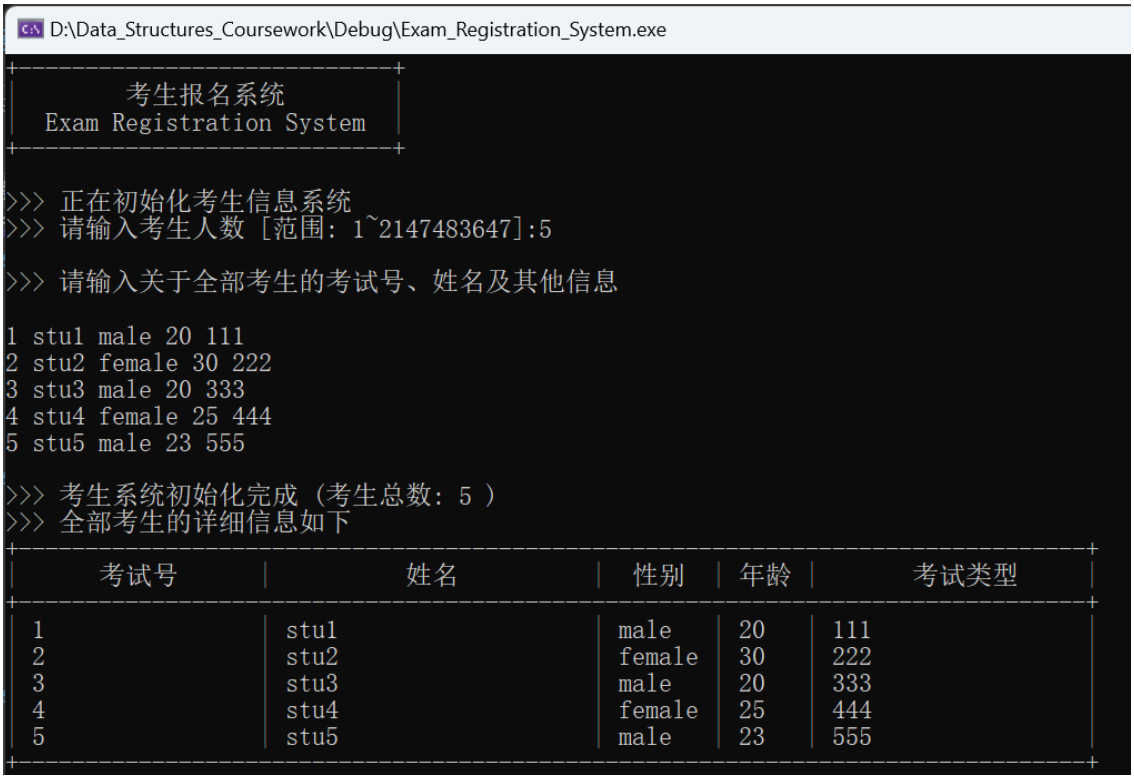


图 4.1 建立考生系统测试示例

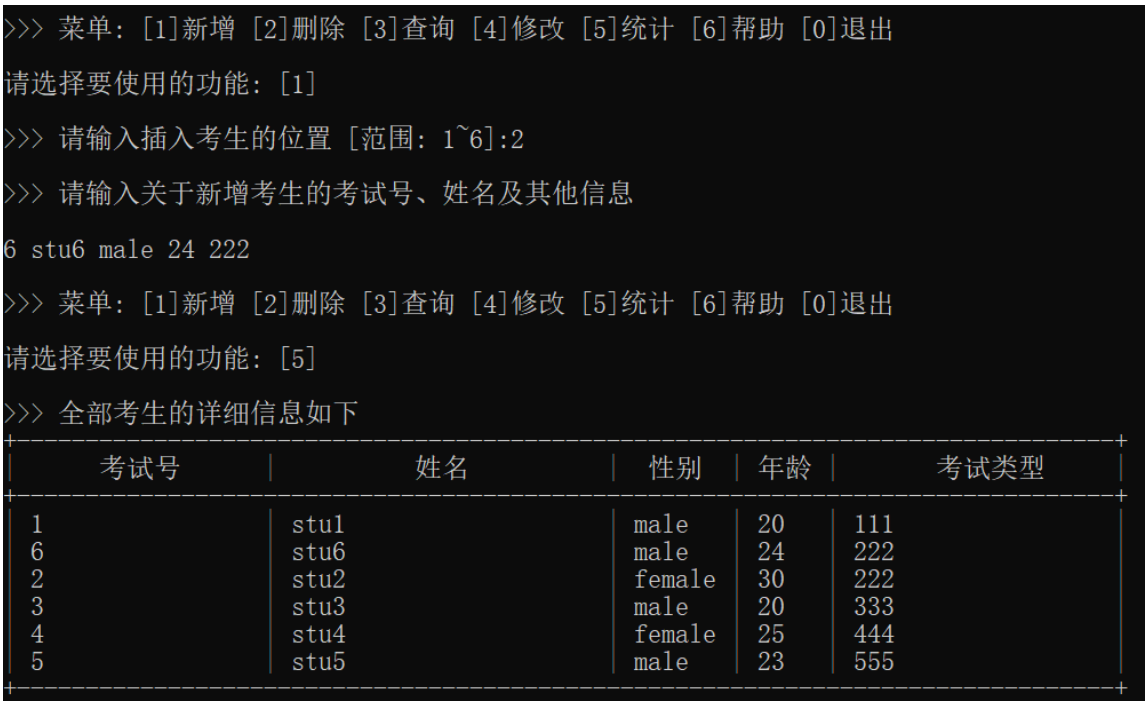


图 4.2 新增考生测试示例

```
>>> 菜单：[1]新增 [2]删除 [3]查询 [4]修改 [5]统计 [6]帮助 [0]退出

请选择要使用的功能：[2]

请输入需删除的考生的考试号:4

>>> 菜单：[1]新增 [2]删除 [3]查询 [4]修改 [5]统计 [6]帮助 [0]退出

请选择要使用的功能：[5]

>>> 全部考生的详细信息如下
+------------------------------------------------------------------------------+
|      考试号      |      姓名      |  性别  | 年龄 |      考试类型      |
+------------------------------------------------------------------------------+
| 1                | stu1           | male   | 20   | 111                |
| 6                | stu6           | male   | 24   | 222                |
| 2                | stu2           | female | 30   | 222                |
| 3                | stu3           | male   | 20   | 333                |
| 5                | stu5           | male   | 23   | 555                |
+------------------------------------------------------------------------------+
```

图 4.3 删除考生测试示例

```
>>> 菜单：[1]新增 [2]删除 [3]查询 [4]修改 [5]统计 [6]帮助 [0]退出

请选择要使用的功能：[3]

请输入需查询的考生的考试号:3
>>> 该考生的详细信息如下
+------------------------------------------------------------------------------+
|      考试号      |      姓名      |  性别  | 年龄 |      考试类型      |
+------------------------------------------------------------------------------+
| 3                | stu3           | male   | 20   | 333                |
+------------------------------------------------------------------------------+
```

图 4.4 查询考生测试示例

```
>>> 菜单：[1]新增 [2]删除 [3]查询 [4]修改 [5]统计 [6]帮助 [0]退出

请选择要使用的功能：[4]

请输入需修改的考生的考试号:6

>>> 请输入关于需修改的考生的考试号、姓名及其他信息

10 stu7 female 23 555

>>> 菜单：[1]新增 [2]删除 [3]查询 [4]修改 [5]统计 [6]帮助 [0]退出

请选择要使用的功能：[5]

>>> 全部考生的详细信息如下
+------------------------------------------------------------------------------+
|      考试号      |      姓名      |  性别  | 年龄 |      考试类型      |
+------------------------------------------------------------------------------+
| 1                | stu1           | male   | 20   | 111                |
| 10               | stu7           | female | 23   | 555                |
| 2                | stu2           | female | 30   | 222                |
| 3                | stu3           | male   | 20   | 333                |
| 5                | stu5           | male   | 23   | 555                |
+------------------------------------------------------------------------------+
```

图 4.5 修改考生信息测试示例

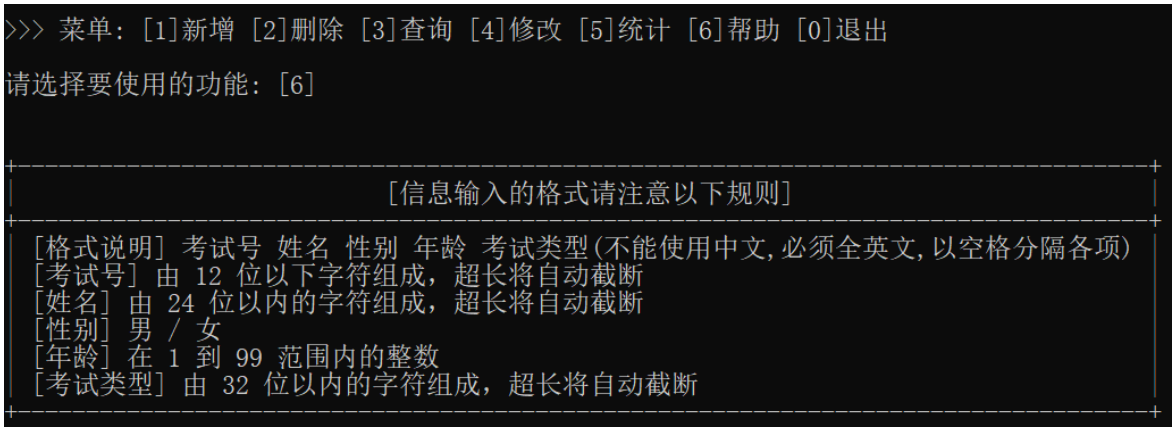


图 4.6 系统输入格式测试示例

5 集成开发环境与编译运行环境

Windows 系统：Windows 11 x64

Windows 集成开发环境：Microsoft Visual Studio 2022 (Release 模式)

Windows 编译运行环境：本项目适用于 x86 架构和 x64 架构