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：

(1)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；

(2)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；

(3)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；

(4)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

|  |
| --- |
|  |

图4.3 删除考生测试示例

|  |
| --- |
|  |

图4.4 查询考生测试示例

|  |
| --- |
|  |

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