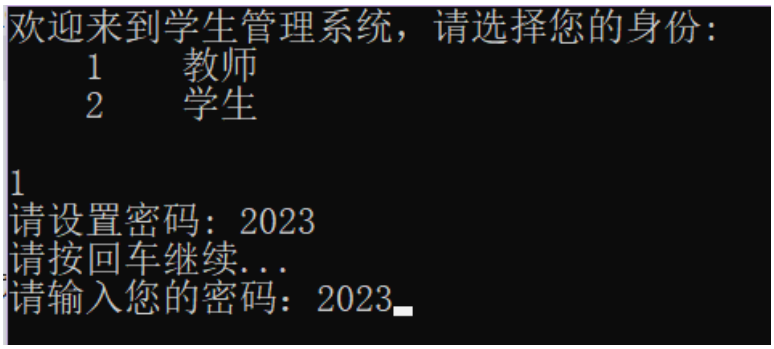
一、学生管理系统使用手册

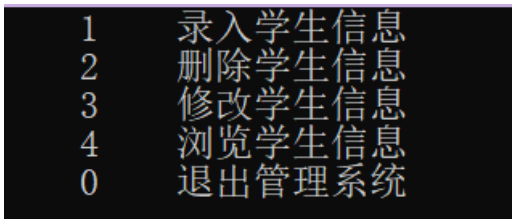
此学生管理系统分为教师端与学生端，教师端可增添、删除、修改与查看学生信息，并可查看各分数段人数及平均分。学生端可按学号或姓名查看自己的成绩与排名，也可查看各分数段人数及平均分，具体使用流程如下：

教师端

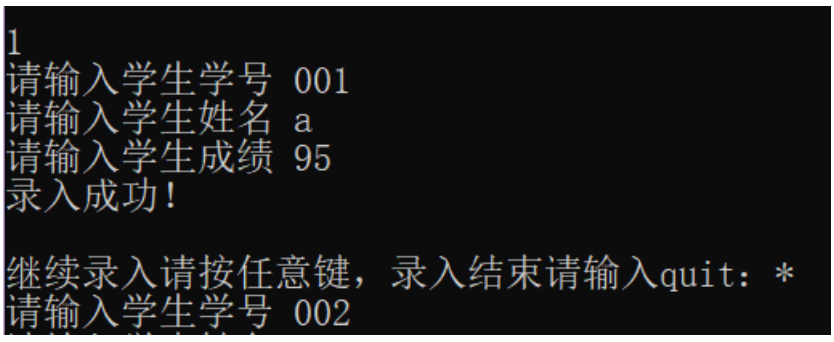
首先需教师使用，教师选择身份并按提示设置密码，如下所示：



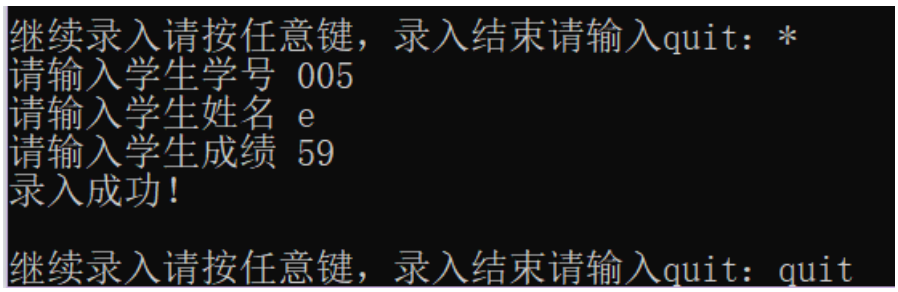
若密码正确，则会进入下一个界面：



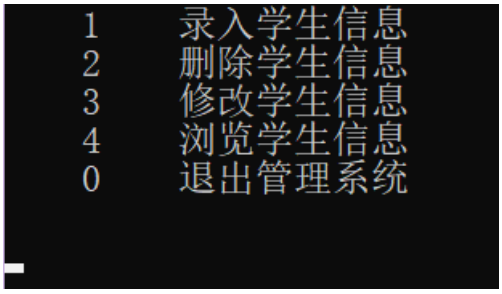
教师可先选择录入学生信息，并按提示录入，最后以quit结束，如下所示：



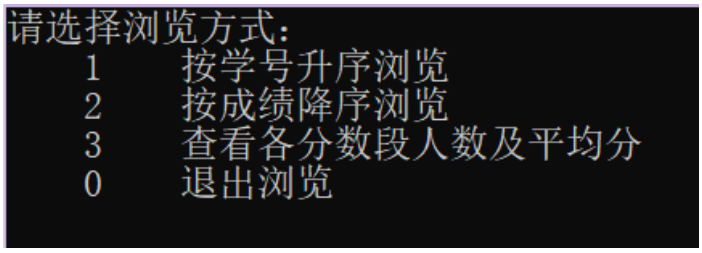
...



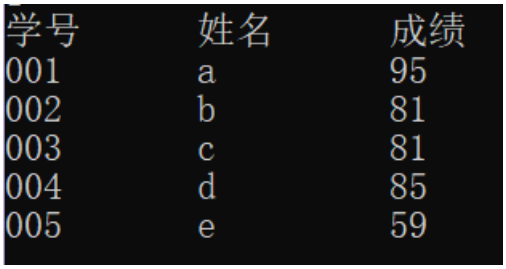
上面的示例共录入了五个学生的信息，当用户录入结束时，命令行会提示“ 录入结束！"并跳转到下个界面：

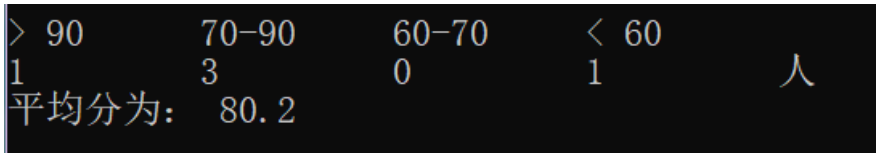


教师可以选择浏览学生信息，此时会跳转到下个界面，为满足老师不同的浏览需求，我们仍提供了以下几种浏览方式：

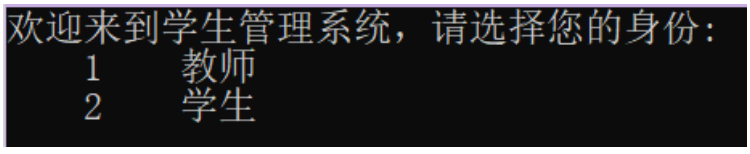


若选择按学号升序浏览并查看各分数段人数，结果分别如下：



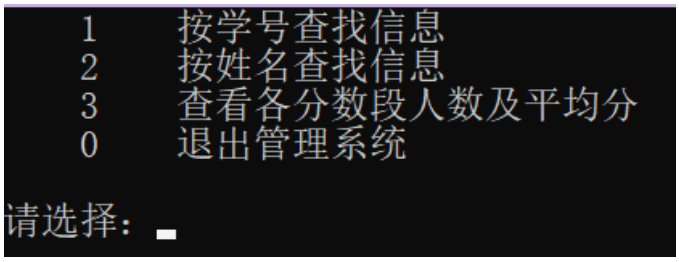


当然老师也可选择删除学生信息和修改学生信息，操作与录入类似，按提示输入即可，在这里就不赘述了。若老师想要退出浏览或退出管理系统输入0即可，退出后会回到主界面。

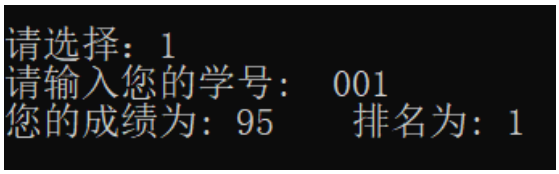


学生端

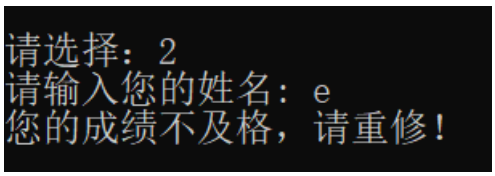
当用户变为学生时，学生也需先选择身份。若有学生选择教师身份，需要先输入密码，密码错误3次将自动返回主界面；若学生选择学生身份，则会进入下一界面，如下所示：



学生可选择按学号查找信息，选择后会出现自己的成绩及排名，如下所示：



当然学生也可按照姓名查找信息，若成绩不及格将不会显示成绩并提醒学生重修，如下所示：



同时学生也可查看各分数段人数。若查询结束，则可选择退出管理系统，回到主界面。

二、代码实现

项目程序共分为四个文件，各文件基本内容及其包含的函数的作用分别如下：

```
//list.h -- 创建单链表，定义链表类成员函数
template <class T>          //引入模板
struct Node {
    Node* next;
    T val;
};

template <class T>
class List {
private:
    Node<T>* first;
    void removeAll();       // 移除链表所有元素
    void copyhelp(const Node<T>*);
    void copyFrom(const List& l);
public:
    bool isEmpty() const;  //判断链表是否为空，为空则返回true,否则返回false
    void insert(T val);    //将val插入到链表中，用first指向新插入的元素
    T removeFirst();       //从链表中移除第一个Node并返回其val值
    Node<T>* returnFirst();             //返回first
```

```
    List();                                 // constructor
    List(const List& l);                    // copy constructor
    List& operator=(const List& l);         // assignment operator
    ~List();                                // destructor
};
...
```

```
//stulist.h -- student结构声明与stulist类定义
#include <iostream>
#include <string>
#include "list.h"

struct student{
    std::string id;
    std::string name;
    int rank = 0;
    double score;
    ...
};

class stulist
{
private:
    List<student> list;           //定义了一个链表，变量类型为student
    int len = 0;                  //表示链表的长度
public:
    void print_list();            //打印list
    void add();                   //增加学生信息
    void dele();                  //删除学生信息
    void modify();                //修改学生信息
    Node<student>* id_find(const std::string);  //判断学号是否存在，若存在则返回在list里的位置(实则是返回指向该学号所在的Node的指针
    Node<student>* name_find(const std::string); //判断姓名是否存在，若存在则返回在list里的位置
    void id_find_(); //按照学号查找信息(若成绩不及格则不显示成绩并提醒重修）
    void name_find();             //按照姓名查找信息
    void score_sort();            //将成绩从高到低排序
    void id_sort();               //将学号从低到高排序
    void score_count();           //统计各分数段人数，计算平均分
    void rank();                  //解决学生有相同分数出现排名不同的情况
};
```

```
//stulist.cpp -- stulist类成员函数的实现
#include <iostream>
#include <cstring>
#include <iomanip>
#include "stulist.h"

void print_list(){
    ...
}
...
```

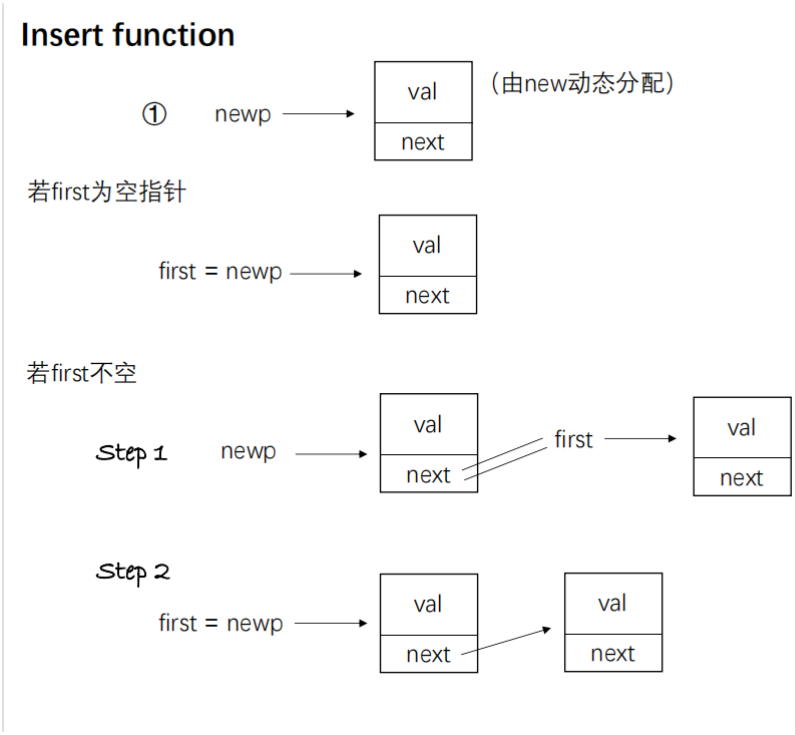
```
//main.cpp -- 调用stulist类的函数的成员函数，负责输入输出
#include<iostream>
#include "stulist.h"
#include <windows.h>
#include<string>

int main(){
    ...
}
```

其中利用了“链表”这一数据结构，思想主要体现在插入和删除元素的过程中， 所以我将集中介绍list.h文件中的insert和removeFirst函数与stulist.cpp文件中的add和dele函数。首先来看insert函数：

```
template <class T>
void List<T>::insert(T val){
    Node<T>* newP = new Node<T>;
    newP->val = val;
    newP->next = nullptr;
    if (!first){
        first = newP;
    }
    else{
        newP->next = first;
        first = newP;
    }
```

可以看出，若想插入一个T型的变量val（之后均用student替代T），首先申请一块动态内存储存Node结构变量，并让newp指向它，之后根据first是否为空指针进行操作。若first为空指针，则直接让first指向这块动态分配的内存；若first不空，则先将first赋给Node里的next，即next去指向first原先所指对象，然后first再指向新分配的内存。用下图解释或许会更清晰：



再看add函数就会比较容易，add函数中最重要的部分如下：

```
void stulist::add()//增加学生信息
{
    ...
    list.insert(student(id, name, score));
    ...
    len++;
}
```

其中最重要的两行即调用了insert函数，插入了一个新的student变量，并让链表的长度加1。

其次就是removeFirst函数，其主要功能是删除链表的头部：

```
template <class T>
T List<T>::removeFirst(){
    Node<T>* victim = first;
    T value = victim->val;
    first = first->next;
    delete victim;
    return value;
}
```

即让一个临时指针victim指向first正指向的元素，而first指向下一个元素，然后删掉victim所指向的这块动态内存，也即删除了第一个元素。

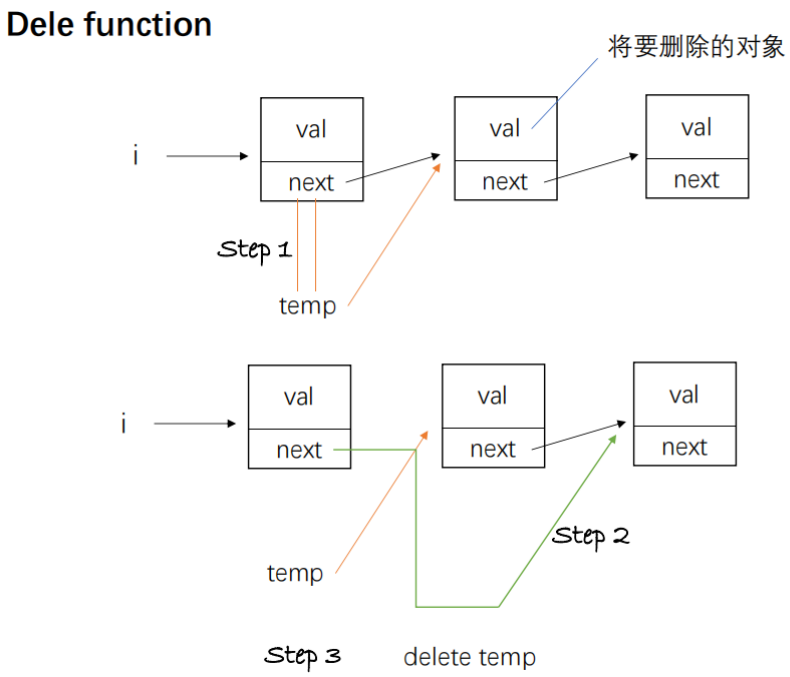
再看dele函数，其主要部分为：

```
void stulist::dele() //删除学生信息
{
    ...
    auto i = list.returnFirst(); //i = first
    if (i->val.id == dele_id) {
        list.removeFirst();
        std::cout << "删除成功" << std::endl;
        len--;
        return;
    }

    for (; i->next != nullptr && i->next->val.id != dele_id; i = i-
>next);

    if (i->next) {
        auto temp = i->next;
        i->next = i->next->next;
        delete temp;
        len--;
        std::cout << "删除成功" << std::endl;
    }
    else
        std::cout << "没有此学号的学生！" << std::endl;
    return;
}
```

dele函数主要运用了removeFirst函数的思想：若要删除的是链表头部元素，则直接调用removeFirst函数；若想要删除链表中间元素，则利用同样的思想，即如下图所示：



除了这几个函数之外，其余的函数基本利用循环或条件就可以实现，在此不再赘述，可根据其实现的功能进行理解。

三、程序优缺点分析

优点

·较传统的学生管理系统新增了教师和学生的身份选择，分隔开教师与学生的使用权限。比如只能由教师进行增删改并查看所有学生信息，而学生只能查询自己的信息

·使用了单链表，较传统的数组储存方式而言，解决了大小限制，删除元素效率更高，且利用动态存储增大了内存空间的利用率

缺点

·只可用于单科目的学生信息管理，希望之后可以实现多科目管理

·无法从文件中读取数据，使得老师录入学生信息的工作量较大

·在处理输入输出时运用了goto语句，应尽量用while语句实现

·为确保学生只能查看自己的信息，应增加学生密码设置功能，考虑可以在student结构里新增code变量