华南师范大学本科生实验报告

姓名 邓智超　　学号 20192121026

院系 计算机学院 专业 计算机科学与技术（师范）

年级 19 级　　班级 1 班

实验时间　2020 年 10 月 13 日

实验名称 学生健康情况管理系统

课程名称 数据结构实验

华南师范大学教务处编印

华南师范大学本科生实验报告

实验课程：数据结构实验课

实验名称：学生健康情况管理系统

一、实验内容

实现学生健康情况管理的几个操作功能（新建、插入、删除、从文件读取、写入文件和查询、屏幕输出等功能）。健康表中学生的信息有学号、姓名、出生日期、性别、身体状况等。

● 必做内容

1. 利用链式存储结构来实现
2. 系统的菜单功能项如下：

1------新建学生健康表

2------向学生健康表添加新的学生信息

3------向学生健康表插入新的学生信息（按位置号来描述插入点）

4------在健康表删除指定学生的信息（按学号操作）

5------为某个学生修改身体状况信息（按学号操作）

6------按学生的学号排序并显示结果

7------在健康表中查询学生信息（按学生学号来进行查找）

8------在屏幕中输出全部学生信息

9------从文件中读取所有学生健康表信息

10------向文件写入所有学生健康表信息

11-----退出

● 选做内容

用 MFC 的单文档窗口和菜单设计界面。

● 已完成内容

必做部分：

本次实验中，我已经在 Visual Studio 平台使用 c++语言完成必做部分的全部

内容，并且对用户交互界面做了一定的优化，使得控制台界面看起来更加简洁舒适。此外，增加了对用户输入合理性的确认。如用户添加的信息是否已经存在，用户查找的目标是否存在等等。所有功能可正常使用且具有一定的健壮性。具体实现过程请查看实验文档部分。

选做部分：

已通过 QT 平台完成了窗口和菜单设计界面。具体的按钮函数已经设计完成，不过最后的组装还需要一段时间。目前（截至 2020/10/26）的界面可以实现“退出”功能，其余按钮可见，但是点击没有反应。具体实现思路请查看实验文档部分。

2020 年 11 月 8 日补充：

已经完成了基于 Qt 平台的学生健康管理系统，实现了图形化的系统。按照需求，该系统可以实现创建表单，添加，删除，修改，插入，查找，排序，输出，读取，写入的功能。我还对界面的切换做出了优化，使得用户在选择不同的功能时，界面的转换更加流畅，（原理上是用户每点击一个功能按钮，就切换到那个功能对应的界面，我将每一个界面的大小、按钮位置等设计成一样，只对部分内容进行修改，这样切换的时候看起来更自然）。接下来将在实验文档部分具体说明实现方式。

目前（截至 2020 年 11 月 8 日）两个系统的缺点在于，写入信息时使用的是 ascll 文件，这就涉及到信息安全的问题了，关于此，将在后续的版本继续改进

二、实验目的

1. 进一步熟悉和掌握 VC 环境下的编译、调试和执行的方法及步骤。
2. 熟悉线性表链式存储的实现方式及其应用。

三、实验文档：

必做部分：

实现思路

学生健康管理系统共包括 student.h studentData.h student.cpp studentData.cpp main.cpp 五个文件，其中定义了两个类（student 类和 studentData

类），student 类用于存放单个学生对象的信息，以及一些设置/获取学生信息的函数构成（具体的后面会提到），studentData 类用于构建链表以及各种对链表的操作实现系统的功能（添加，删除等等）。main 函数用于显示菜单、接受用户指令并调用对应的函数，以统筹各项操作。

student 类声明

私有成员用于存储学生数据包括 name(姓名)，number(学号)，gender(性别)，dateOfBirth(出生日期)和 healthCondition(健康状况)，这些属性都声明为 string 类。公有成员有：用于指向链表下一节点的指针 student*next、构造函数(作用是将学生的属性都初始化为"")、一堆返回信息的函数(getName()、getDateOfBirth()、getGender()、getHealthCondition()和 getNumber，实现方法很简单，返回对应属性即可，在此不做详细说明)、addStudent()函数（注意，这并不是真的将学生加入链表，而是接受输入，将学生的数据修改为用户需要的内容。加入链表的操作将在 studentData 类的函数中实现，这样命名是为了使得类似功能的函数名相同）、重载<<和 exchange()函数（交换两个对象的数据，运用选择排序的时候便于交换数据）。

student 类声明代码如下：

```
#pragma once
#ifndef _student_h_
#define _student_h_
#include<string>
using namespace std;
//
//以下是 student 类的定义，其主要作用是存储每一个学生的各项数据
//
class student
{
private:
    string name;//姓名
    string number;//学号
```

```
    string dateOfBirth;//出生日期
    string gender;//性别
    string healthCondition;//健康状况
public:
    student* next;//指向下一节点
    student();//构造函数，初始化信息
    string getName();//返回姓名
    string getDateOfBirth();//返回出生日期
    string getGender();//返回性别
    string getHealthCondition();//返回健康状况
    string getNumber();//返回学号
    //将某对象的属性赋值形参分别代表姓名，学号，性别，出生日期，健康状况
    bool addStudent(string n, string num, string g,string dOB, string hC);
    friend ostream& operator <<(ostream& out, student& student1);//重载<<
    void exchange(student& w2);//交换两个对象的数据，运用选择排序的时候便于交换数据
};
#endif
```

student 类实现

接下来将介绍 addStudent()，重载<<运算符，exchange()的实现方法。

addStudent()

函数原型：

```
bool addStudent(string n, string num, string g,string dOB, string hC);
```

实现方式：

此函数的功能是修改类对象的数据。将 name 赋值为 n，number 赋值为 num，gender 赋值为 g，dateOfBirth 赋值为 dOB，healthCondition 赋值为 hC。若添加成功，则返回逻辑值 1

代码：

```
//"添加"成员，将对象的各个属性赋值为函数参数的内容
bool student::addStudent(string n, string num, string g, string dOB, string hC)
{
    name=n;
    number = num;
    dateOfBirth = dOB;
    gender = g;
    healthCondition = hC;
    return 1;
}
```

重载<<运算符

函数原型：

```
friend ostream& operator <<(ostream& out, student& student1);//重载<<
```

实现方式：

此函数的功能是对 student 类重载<<运算符。按照一定的字长，依次输出 student 类对象的 name，number，gender，dateOfBirth，healthCondition 属性。经过测试，这五个属性的最佳字长为 8，13，6，16，10（setw()函数需要头文件 iomanip）。

代码：

```
//对 student 类重载<<运算符，便于输出
ostream& operator <<(ostream& out, student& student1)//重载<<
{
    cout << setw(8) << student1.name;
    cout << setw(13) << student1.number;
    cout << setw(6) << student1.gender;
    cout << setw(16) << student1.dateOfBirth;
    cout << setw(10) << student1.healthCondition;
    return out;
```

}

exchange()

函数原型：

void exchange(student& w2);

实现方式：

此函数的功能是实现两个类对象数据的交换。声明一个 string 类变量 temp 用于临时存储，实现两个对象（this 和 w2）的 name，number，gender，dateOfBirth，healthCondition 的互换。因为是 string 类，所以赋值直接用=即可。

代码：

```
//交换两个对象的数据
void student::exchange(student& w2)
{
    string temp1;
    temp1 = this->name;//交换姓名
    this->name = w2.name;
    w2.name = temp1;
    temp1 = this->number;//交换学号
    this->number = w2.number;
    w2.number = temp1;
    temp1 = this->gender;//交换性别
    this->gender = w2.gender;
    w2.gender = temp1;
    temp1 = this->dateOfBirth;//交换出生日期
    this->dateOfBirth = w2.dateOfBirth;
    w2.dateOfBirth = temp1;
    temp1 = this->healthCondition;//交换健康状况
    this->healthCondition = w2.healthCondition;
    w2.healthCondition = temp1;
```

```
}
```

综上，student 类的所有实现代码如下：

```
#include"student.h"
#include<iostream>
#include<iomanip>
#include<string>
using namespace std;

//构造函数，将所有属性初始化为""
student::student()
{
	name = "";
	number = "";
	gender = "";
	dateOfBirth = "";
	healthCondition = "";
}

//返回姓名
string student::getName()
{
	return name;
}

//返回出生日期
string student::getDateOfBirth()
{
	return dateOfBirth;
```

```
}

//返回性别
string student::getGender()
{
	return gender;
}

//返回健康状况
string student::getHealthCondition()
{
	return healthCondition;
}

//返回学号
string student::getNumber()
{
	return number;
}

////"添加"成员，将对象的各个属性赋值为函数参数的内容
bool student::addStudent(string n, string num, string g, string dOB, string hC)
{
	name=n;
	number = num;
	dateOfBirth = dOB;
	gender = g;
	healthCondition = hC;
	return 1;
```

```
}

//对 student 类重载<<运算符，便于输出
ostream& operator <<(ostream& out, student& student1)//重载<<
{
	cout << setw(8) << student1.name;
	cout << setw(13) << student1.number;
	cout << setw(6) << student1.gender;
	cout << setw(16) << student1.dateOfBirth;
	cout << setw(10) << student1.healthCondition;
	return out;
}

//交换两个对象的数据
void student::exchange(student& w2)
{
	string temp1;
	temp1 = this->name;//交换姓名
	this->name = w2.name;
	w2.name = temp1;
	temp1 = this->number;//交换学号
	this->number = w2.number;
	w2.number = temp1;
	temp1 = this->gender;//交换性别
	this->gender = w2.gender;
	w2.gender = temp1;
	temp1 = this->dateOfBirth;//交换出生日期
	this->dateOfBirth = w2.dateOfBirth;
	w2.dateOfBirth = temp1;
```

```
    temp1 = this->healthCondition;//交换健康状况
    this->healthCondition = w2.healthCondition;
    w2.healthCondition = temp1;
}
```

studentData 类声明

studentData 类用于将学生信息构建成链表，以及进行各种链表操作。私有属性只有一个，那就是用于存储链表首地址的指针 student*first。私有成员包括：string 类对象 fileName（用于存储表单的名字）、establish()函数、readData()函数、writeData()函数、add()函数、del()函数、change()函数、searchNum()函数，searchName()函数，list()函数、rank()函数、insert 函数()，分别实现新建健康表、读取文件、写入文件、添加信息、删除信息、修改信息、查找信息、输出所有信息、排序、插入信息的功能。在接下来的内容会逐个说明每个函数的实现方法。

studentData 类声明代码如下：

```
#ifndef _studentData_h_
#define _studentData_h_
#include"student.h"
//
//接下来是 studentData 类的定义，其作用是通过链表形式组建文件数据
//并对数据进行添加、删除、插入、查找、读取、写入等等操作
//
class studentData
{
private:
    student* first;
public:
    string fileName;//文件名
    void establish();//新建健康表
    void readData();//  读取文件，生成链表
```

```
    void writeData();//写入文件，更新内容
    student* add();//添加信息
    bool del();//删除信息
    student* change();//修改信息
    //按学号查找信息，flag 为 0 时返回目标节点地址，flag 为 1 时，返回前一个结点地址
    student* searchNum(string n,int flag);
    student* searchName(string n);//按姓名查找，返回目标节点地址
    bool list();//输出所有信息
    bool rank();//排序
    ~studentData();//析构函数，写入文件，更新内容，释放链表空间
    bool insert();//插入
};
#endif
```

studentData 类实现

接下来将逐个介绍 establish()函数、readData()函数、writeData()函数、add()函数、del()函数、change()函数、searchNum()函数，searchName()函数，list()函数、rank()函数、insert 函数()的实现方法（根据需要会适度调整说明顺序，并不是严格按照以上顺序的）。

establish()

函数原型：

```
void studentData::establish()
```

实现方式：

此函数的功能是新建一个表单。提示用户输入想要创建的表单名称，将用户的输入赋值给 fileName 属性，之后将 fileName 字符串之后加上字符串常量".dat"，这样 fileName 就是之后要写入的文件夹名称了。最后，提示用户建立成功即可。（说明：此处并没有建立文件夹，文件夹的创建是在 writeData()函数中实现的，也就是

说，用户创建文件后，需要选择写入功能才能保存文件，这样的设定与目前人们的使用习惯是一致的）

代码：

```
//建立新的文件表，根据用户输入，确定文件名
void studentData::establish()
{
    cout << "请输入健康表名称：" << endl;
    cin >> fileName;
    fileName = fileName + ".dat";
    cout << "建立成功!! " << endl << endl;
}
```

list()

函数原型：

bool studentData::list()

实现方式：

此函数的功能是输出链表的所有数据。为了让输出的每一列有一个标题，故又声明了一个 title()函数，其作用就是输出“姓名”“学号”“性别”“出生日期”“健康状况”。根据尝试，这五个字符串的最佳字长为 8，13，6，16，10（setw()函数需要头文件 iomanip）。在 list()函数中，先判断链表是否为空（first==0），若为空，就输出提示信息，若不为空，就声明一个指向 student 类的指针 current，初值为 first，遍历整个链表，对于 current 的每一个值，都使用语句 cout<<*current<<endl;（之前已经对 student 类的<<运算符进行重载），循环终止的条件是 current==0。

代码：

```
//输出列表时作为标题
void title()
{
    cout << setw(8) << "姓名";
    cout << setw(13) << "学号";
    cout << setw(6) << "性别";
```

```
	cout << setw(16) << "出生日期";
	cout << setw(10) << "健康状况";
	cout << endl;
}

//输出所有信息
//若链表为空，报错
//此处已经对 student 类重载了<<操作符，直接用<<输出对象信息即可
bool studentData::list()
{
	cout << endl << endl;
	if (first == 0)cout << "ERROR：信息库为空，无相关信息！" << endl;

	else
	{
		title();
		student* current = first;
		while (current != 0)
		{
			cout << *current << endl;
			current = current->next;
		}
	}
	cout << endl << endl;
	return 1;
}
```

readData()

函数原型：

void studentData::readData()

实现方式：

此函数的功能是实现读取指定文件名的文件数据，并将数据建立成链表。声明一个 string 类的对象 fileName1，提示用户输入想要读取的文件名，将用户的输入赋值给 fileName1，之后，在 fileName1 后加上字符串常量".dat"，这样 fileName1 就是要打开的文件了。声明 fstream 类对象 file，以读的方式打开文件，若打开失败，则输出提示。之后，声明 string 类对象 name，number，gender，dateOfBirth，healthCondition，从文件中一次读取 5 个字符串并按顺序赋值给这 5 个对象，然后

动态申请一个 student 类对象，其地址为 p，调用 p->addStudent()函数将读入的数据赋值给类对象，然后用头插入的方式将新节点插入链表。循环结束的条件是!file。若读取后链表为空，就输出提示，否则，输出提示读取成功并调用 list()函数将读取到的数据输出。

代码：

```
void studentData::readData()
{
	fstream file;
	string fileName1;
	cout << "请输入要读取的文件名:" << endl;
	cin >> fileName1;
	file.open(fileName1 + ".dat", ios::in);
	string name;
	string number;
	string gender;
	string dateOfBirth;
	string healthCondition;
	while (1)
	{
		file >> name >> number >> gender >> dateOfBirth >> healthCondition;
		if (!file)break;
		student* p = new student;
		p->addStudent(name, number, gender, dateOfBirth, healthCondition);
		p->next = first;//头插入方式，可以避免判断链表是否为空
		first = p;
	}
	file.close();
	if (first == 0)
	{
```

```
            cout << "ERROR：该文件为空!! " << endl << endl;
            return;
        }
        else cout << "读取成功!! " << endl << endl;
        list();//输出所有信息
}
```

writeData()

函数原型：

```
    void studentData::writeData()
```

实现方式：

此函数的功能是将链表的数据写入指定文件。若在调用此函数前，用户未输入表单名，则提示用户输入表单名，并赋值给 fileName，在 fileName 之后加上字符串常量”.dat”作为写入的文件名。若链表为空，则输出提示。若不为空，则声明 fstream 类对象 file，打开文件以写的方式打开文件（此文件会覆盖之前的同名文件以达到刷新数据的目的）。声明一个指向 student 类的指针 current 并赋初为 first，之后开始遍历链表，每循环一次，就将 current 所在节点的 name，number，gender，dateOfBirth，healthCondition 写入文件，注意每写入一个字符串，就再写入一个空格，便于之后的读入。循环结束的条件是 current==0。最后，输出信息提示用户写入成功。

代码：

```
//写入文件
//若 fileName 为空，则先让用户输入要存储的文件名
//若链表为空，报错
//按姓名，学号，性别，出生日期，健康状况写入文件
void studentData::writeData()
{
    if (fileName == "")
```

```
    {
        cout << "请输入要存入的文件名：" << endl;
        cin >> fileName;
        fileName = fileName + ".dat";
    }
    student* current = first;
    if (first == 0)
    {
        cout << "ERROR：链表为空，无法写入，请输入数据!! " << endl;
        return;
    }
    fstream file;
    file.open(fileName, ios::out);
    while (current != 0)
    {
        file << current->getName() << " " << current->getNumber() << " " <<
current->getGender() << " "
            << current->getDateOfBirth() << " " << current->getHealthCondition() << " ";
        current = current->next;
    }
    file.close();
    cout << "写入数据成功！!" << endl << endl;
}
```

searchNum()

函数原型：

student* studentData::searchNum(string n, int flag)

实现方式：

此函数的共能按照学号查找节点位置，返回值是节点地址。若 flag 是 0，返回

目标节点地址，若 flag 是 1，返回目标节点前一个节点的地址，这样设计是便于插入和删除，因为插入和删除都涉及到前一个节点地址的问题。声明指向 student 类的指针 current 并赋初值为 first，之后开始遍历链表，若遍历后 current 为 0，则未找到相关信息，返回 0。若找到，则输出该学生的信息。

代码：

```
//按照学号查找结点位置，返回值是地址
//若 flag==0，返回目标节点地址
//若 flag==1，返回目标节点前一个节点的地址，便于插入和删除
student* studentData::searchNum(string n, int flag)
{
	student* current = first;
	student* prenode = 0;
	while (current != 0 && current->getNumber() != n)
	{
		prenode = current;
		current = current->next;
	}
	if (current == 0)//没有查到信息
	{
		return 0;
	}
	else if (flag == 0 || current == first)return current;//返回目标节点地址
	else return prenode;//返回目标节点地址的前一个地址
}
```

searchName()

函数原型：

```
student* studentData::searchName(string n)
```

实现方式：

此函数用于辅助判断用户在添加信息时，学生的姓名是否重复。具体实现方式与 searchNum()类似，此处不再复述。不过，searchName()函数无 flag 形参,返回值就是目标节点地址。

代码：

```
//按照姓名查找，直接返回目标节点地址
//实现思路与学号查找类似
student* studentData::searchName(string n)
{

    student* current = first;
    student* prenode = 0;
    while (current != 0 && current->getName() != n)
    {
        prenode = current;
        current = current->next;
    }
    if (current == 0)//未查到信息
    {
        return 0;
    }
    return current;//返回目标节点地址
}
```

add()

函数原型：

```
student* studentData::add()
```

实现方式：

该函数的功能是实现添加节点。考虑直接将新节点作为头节点（即采用头插入方式）。先获取用户的信息输入，之后调用 searchName()和 searchNum()函数检查姓名和学号有没有在链表内重复（若 searchName()和 searchNum()函数的返回值都是 0，说明没有重复），若有重复，则输出提示信息并结束函数，若没有重复，则新申请一个结点，并将用户输入赋值给类对象，之后采用头插入的方式插入节点，最后输出添加成功的信息。

代码：

```
//采用头插入方式
//插入前先判断姓名，学号是否重复
student* studentData::add()//添加信息
{
    string name, number,gender, dateOfBirth, healthCondition;
    cout << "请分别输入学生的姓名，学号，性别，出生日期，健康状况" << endl;
    cin >> name >> number >> gender >> dateOfBirth >> healthCondition;
    //判断姓名是否重复
    if (searchName(name) != 0)
    {
        cout << "ERROR：姓名为" << name << "的学生已经存在！" << endl << endl << endl;
        return first;
    }
    //判断学号是否重复
    else if (searchNum(number,0) != 0)
    {
        cout << "ERROR：学号为" << number << "的学生已经存在！" << endl << endl << endl;
        return first;
    }
    student* p = new student;
    p->addStudent(name, number, gender, dateOfBirth, healthCondition);//输入结点信息
    //插入节点
```

```
    p->next = first;
    first = p;
    cout << "添加成功!! " << endl << endl;
    return first;
}
```

insert()

函数原型：

```
bool studentData::insert()
```

实现方式：

该函数的功能是实现插入新的节点到指定位置。首先声明一个 current 指针并赋初值为 first，声明 int 型变量 address 接受用户输入的位置信息，之后需要将 current 指针移动到指向第 address-1 节点。在此需要进行特殊情况的判断，若 address 是 1，则直接采用头插入插入一个节点即可，若 address-1 比链表的长度还大，就需要输出提示信息（默认将节点插在链表最后）。移动完 current 指针之后，插入节点即可。对于这个操作，还需要判断新数据的姓名和学号数据是否与现有数据重复，若重复则输出提示信息。

代码：

```
//按照用户输入的位置，将 current 指针进行相应的移动，使得指针指向要插入位置的前一个指针
//若用户向插入的位置的前一个位置没有节点，则输出提醒，并默认插入到最后一个位置！
//插入信息时，要对信息是否重复进行判断
bool studentData::insert()
{
    int address;
    cout << "你想在第几个位置插入信息？ " << endl;
    cin >> address;
    student* current=first;
```

```
    //将指针进行移动
    for (int i = 1; i <= address-2; i++)
    {
        current = current->next;//将指针移动到插入位置
        if (current->next == 0&&i!=address-2)
        {
            cout << "WARNING：该信息表的第" << address - 1 << "个位置没有信息，故不能在第"
                << address << "个位置插入信息，默认在最后位置插入！" << endl;
            break;
        }

    }
    //输入信息
    string name, number, gender, dateOfBirth, healthCondition;
    cout << "请分别输入学生的姓名，学号，性别，出生日期，健康状况" << endl;
    cin >> name >> number >> gender >> dateOfBirth >> healthCondition;
    //判断姓名是否重复
    if (searchName(name) != 0)
    {
        cout << "ERROR：姓名为" << name << "的学生已经存在！" << endl << endl << endl;
        return 0;
    }
    //判断学号是否重复
    else if (searchNum(number, 0) != 0)
    {
        cout << "ERROR：" << number << "的学生已经存在！" << endl << endl << endl;
        return 0;
    }
```

```
    student* p = new student;
    p->addStudent(name, number, gender, dateOfBirth, healthCondition);//赋值
    //插入位置是第一个节点
    if (address == 1)
    {
        p->next = first;
        first = p;
    }
    //其他情况
    else
    {
        p->next = current->next;
        current->next = p;
    }
    cout << "添加成功!! " << endl << endl;
    return 1;
}
```

del()

函数原型：

```
bool studentData::del()
```

实现方式：

此函数的功能是实现删除某一学生的信息。若此时链表为空或链表内无目标信息，则输出提示信息。此外，分两种情况讨论，若要删除的目标节点是头节点，则需要将 first 先赋值给临时变量 temp，之后将 first 赋值为 first->next，然后释放 temp 所指向的节点空间；若要删除的节点不是头节点，则先需要调用 searchNum()函数获取目标节点的前一节点的地址，并将前一节点指向目标节点的后一节点，之后释放目标节点的空间。

代码：

```
bool studentData::del()//删除信息
{
	cout << "请输入要删除者的学号：" << endl;
	string number1;
	cin >> number1;
	student* p = searchNum(number1, 1);//返回前一结点地址
	if (p == 0)
	{
		cout << "ERROR：无相关信息！" << endl << endl;
		return 0;
	}
	else
	{
		//要删除的节点是表头
		if (p == first)
		{
			first = p->next;
			delete p;
		}
		//其他情况
		else
		{
			p->next = (p->next)->next;
			delete p->next;
		}
	}
	cout << "删除成功!！" << endl << endl;
	return 1;
```

```
}
```

change()

函数原型：

```
student* studentData::change()
```

实现方式：

此函数功能是实现修改某一学生的健康状态。先声明一个指向 student 类的指针 current。用户的输入是学号，所以调用 searchNum()函数获取目标节点的地址，并将返回值赋值给 current，若 current 是 0，则说明没有目标信息，这时需要输出提示信息，若 current 不是 0，则将*current 的 healthCondition 属性修改成用户输入的数据即可。

代码：

```
//修改信息
student* studentData::change()
{
    cout << "请输入要修改者的学号：" << endl;
    string number1;
    cin >> number1;
    student* p = searchNum(number1,0);//返回目标结点地址
    if (p == 0)
    {
        cout << "ERROR：无相关信息！" << endl << endl;
        return first;
    }
    else
    {
        string condition;
```

```
            cout << "请输入修改后的学生健康状况：" << endl;
            cin >> condition;
            cout << "信息修改成功！" << endl << endl;
            p->addStudent(p->getName(), p->getNumber(), p->getGender(), p->getDateOfBirth(),
condition);//刷新信息
            return first;
        }
}
```

rank()

函数原型：

bool studentData::rank()

实现方式：

该函数的功能是实现将链表信息按照学号大小排序，之后输出排序后的所有信息。若链表为空，则不进行排序，直接输出提示信息。若链表不为空，考虑采用选择排序的方法将链表进行排序，涉及到交换两个节点的信息时，可以调用exchange()函数（函数声明之前已经提到）。排序完成后，输出提示信息，并调用list()函数将链表内容输出。

代码：

```
//按照学号从小到大排序
//用选择排序的方法
//已经定义了 exchange 函数，可以用于交换两个节点的内容
bool studentData::rank()//排序
{
    if (first == 0)
    {
        cout << "ERROR：信息库为空，无法排序，请输入数据！" << endl;
        return 0;
    }
    student* p=0;
```

```
    student* q=0;
    //选择排序
    for(p=first;p!=0;p=p->next)
        for (q = p->next; q != 0; q = q->next)
            if (q->getNumber() < p->getNumber())
                p->exchange(*q);//交换数据
    cout << "已经按照学号大小排序成功!! " << endl << endl;
    list();//输出信息
    return 1;
}
```

析构函数

函数原型：

studentData::~studentData()

实现方式：

该函数的功能是析构链表，释放存储空间。声明一个指向 student 类的 current 指针，配合 first 指针的移动，即可实现逐个析构节点的目标。

代码：

```
//释放链表空间
studentData::~studentData()
{
    student* current;
    while (first != 0)
    {
        current = first;
        first = first->next;
        delete current;
    }
```

```
}
```

综上，studentData 类的实现代码如下：

```
#include"studentData.h"
#include<fstream>
#include<iostream>
#include<iomanip>
using namespace std;

//建立新的文件表，根据用户输入，确定文件名
void studentData::establish()
{
	cout << "请输入健康表名称：" << endl;
	cin >> fileName;
	fileName = fileName + ".dat";
	cout << "建立成功!! " << endl << endl;
}

//读取文件，构建链表
//若文件为空，报错
//每次读入五个字符串，分别赋值给姓名，学号，性别，出生日期，健康状况，之后通过头插入建链
void studentData::readData()
{
	fstream file;
	string fileName1;
	cout << "请输入要读取的文件名:" << endl;
	cin >> fileName1;
	file.open(fileName1 + ".dat", ios::in);
	string name;
```

```
    string number;
    string gender;
    string dateOfBirth;
    string healthCondition;
    while (1)
    {
        file >> name >> number >> gender >> dateOfBirth >> healthCondition;
        if (!file)break;
        student* p = new student;
        p->addStudent(name, number, gender, dateOfBirth, healthCondition);
        p->next = first;//头插入方式，可以避免判断链表是否为空
        first = p;
    }
    file.close();
    if (first == 0)
    {
        cout << "ERROR：该文件为空！！" << endl << endl;
        return;
    }
    else cout << "读取成功！！" << endl << endl;
    list();//输出所有信息，函数实现在后面
}

//写入文件
//若 fileName 为空，则先让用户输入要存储的文件名
//若链表为空，报错
//按姓名，学号，性别，出生日期，健康状况写入文件
void studentData::writeData()
{
```

```
    if (fileName == "")
    {
        cout << "请输入要存入的文件名：" << endl;
        cin >> fileName;
        fileName = fileName + ".dat";
    }
    student* current = first;
    if (first == 0)
    {
        cout << "ERROR：链表为空，无法写入，请输入数据!! " << endl;
        return;
    }
    fstream file;
    file.open(fileName, ios::out);
    while (current != 0)
    {
        file << current->getName() << " " << current->getNumber() << " " <<
current->getGender() << " "
            << current->getDateOfBirth() << " " << current->getHealthCondition() << " ";
        current = current->next;
    }
    file.close();
    cout << "写入数据成功！!" << endl << endl;
}

////输出列表时作为标题
void title()
{
    cout << setw(8) << "姓名";
```

```
	cout << setw(13) << "学号";
	cout << setw(6) << "性别";
	cout << setw(16) << "出生日期";
	cout << setw(10) << "健康状况";
	cout << endl;
}

//输出所有信息
//若链表为空，报错
//此处已经对 student 类重载了<<操作符，直接用<<输出对象信息即可
bool studentData::list()
{
	cout << endl << endl;
	if (first == 0)cout << "ERROR：信息库为空，无相关信息！" << endl;

	else
	{
		title();
		student* current = first;
		while (current != 0)
		{
			cout << *current << endl;
			current = current->next;
		}
	}
	cout << endl << endl;
	return 1;
}
```

```
//按照学号查找结点位置，返回值是地址
//若 flag==0，返回目标节点地址
//若 flag==1，返回目标节点前一个节点的地址，便于插入和删除
student* studentData::searchNum(string n, int flag)
{
	student* current = first;
	student* prenode = 0;
	while (current != 0 && current->getNumber() != n)
	{
		prenode = current;
		current = current->next;
	}
	if (current == 0)//没有查到信息
	{
		return 0;
	}
	else if (flag == 0 || current == first)return current;//返回目标节点地址
	else return prenode;//返回目标节点地址的前一个地址
}

//按照姓名查找，直接返回目标节点地址
//实现思路与学号查找类似
student* studentData::searchName(string n)
{
	student* current = first;
	student* prenode = 0;
	while (current != 0 && current->getName() != n)
	{
		prenode = current;
```

```
            current = current->next;
        }
        if (current == 0)//未查到信息
        {
            return 0;
        }
        return current;//返回目标节点地址
}

//按照用户输入的位置，将 current 指针进行相应的移动，使得指针指向要插入位置的前一个指针
//若用户向插入的位置的前一个位置没有节点，则输出提醒，并默认插入到最后一个位置！
//插入信息时，要对信息是否重复进行判断
bool studentData::insert()
{
        int address;
        cout << "你想在第几个位置插入信息？" << endl;
        cin >> address;
        student* current=first;
        //将指针进行移动
        for (int i = 1; i <= address-2; i++)
        {
            current = current->next;//将指针移动到插入位置
            if (current->next == 0&&i!=address-2)
            {
                cout << "WARNING：该信息表的第" << address - 1 << "个位置没有信息，故不能在第"
                    << address << "个位置插入信息，默认在最后位置插入！" << endl;
                break;
```

```
        }
    }
    //输入信息
    string name, number, gender, dateOfBirth, healthCondition;
    cout << "请分别输入学生的姓名，学号，性别，出生日期，健康状况" << endl;
    cin >> name >> number >> gender >> dateOfBirth >> healthCondition;
    //判断姓名是否重复
    if (searchName(name) != 0)
    {
        cout << "ERROR：姓名为" << name << "的学生已经存在！" << endl << endl << endl;
        return 0;
    }
    //判断学号是否重复
    else if (searchNum(number, 0) != 0)
    {
        cout << "ERROR：" << number << "的学生已经存在！" << endl << endl << endl;
        return 0;
    }
    student* p = new student;
    p->addStudent(name, number, gender, dateOfBirth, healthCondition);//赋值
    //插入位置是第一个节点
    if (address == 1)
    {
        p->next = first;
        first = p;
    }
    //其他情况
    else
    {
```

```
            p->next = current->next;
            current->next = p;
        }
        cout << "添加成功!!  " << endl << endl;
        return 1;
}

//采用头插入方式
//插入前先判断姓名，学号是否重复
student* studentData::add()//添加信息
{
        string name, number,gender, dateOfBirth, healthCondition;
        cout << "请分别输入学生的姓名，学号，性别，出生日期，健康状况" << endl;
        cin >> name >> number >> gender >> dateOfBirth >> healthCondition;
        //判断姓名是否重复
        if (searchName(name) != 0)
        {
            cout << "ERROR：姓名为" << name << "的学生已经存在！" << endl << endl << endl;
            return first;
        }
        //判断学号是否重复
        else if (searchNum(number,0) != 0)
        {
            cout << "ERROR：学号为" << number << "的学生已经存在！" << endl << endl << endl;
            return first;
        }
        student* p = new student;
        p->addStudent(name, number, gender, dateOfBirth, healthCondition);//输入结点信息
        //插入节点
```

```
	p->next = first;
	first = p;
	cout << "添加成功!!" << endl << endl;
	return first;
}

//利用 searchNum 函数返回的前一节点地址，实现删除操作
bool studentData::del()//删除信息
{
	cout << "请输入要删除者的学号：" << endl;
	string number1;
	cin >> number1;
	student* p = searchNum(number1, 1);//返回前一结点地址
	if (p == 0)
	{
		cout << "ERROR：无相关信息！" << endl << endl;
		return 0;
	}
	else
	{
		//要删除的节点是表头
		if (p == first)
		{
			first = p->next;
			delete p;
		}
		//其他情况
		else
		{
```

```
                p->next = (p->next)->next;
                delete p->next;
            }
        }
        cout << "删除成功!! " << endl << endl;
        return 1;
}

//修改信息
student* studentData::change()
{
        cout << "请输入要修改者的学号：" << endl;
        string number1;
        cin >> number1;
        student* p = searchNum(number1,0);//返回目标结点地址
        if (p == 0)
        {
            cout << "ERROR：无相关信息！" << endl << endl;
            return first;
        }
        else
        {
            string condition;
            cout << "请输入修改后的学生健康状况：" << endl;
            cin >> condition;
            cout << "信息修改成功！" << endl << endl;
            p->addStudent(p->getName(), p->getNumber(), p->getGender(), p->getDateOfBirth(),
condition);//刷新信息
            return first;
```

```
    }
}

//按照学号从小到大排序
//用选择排序的方法
//已经定义了 exchange 函数，可以用于交换两个节点的内容
bool studentData::rank()//排序
{
    if (first == 0)
    {
        cout << "ERROR：信息库为空，无法排序，请输入数据！" << endl;
        return 0;
    }
    student* p=0;
    student* q=0;
    //选择排序
    for(p=first;p!=0;p=p->next)
        for (q = p->next; q != 0; q = q->next)
            if (q->getNumber() < p->getNumber())
                p->exchange(*q);//交换数据
    cout << "已经按照学号大小排序成功！！" << endl << endl;
    list();//输出信息
    return 1;
}

//释放链表空间
studentData::~studentData()
{
    student* current;
```

```
    while (first != 0)
    {
        current = first;
        first = first->next;
        delete current;
    }
}
```

main 函数实现

实现方式：

该函数的功能是实现与用户的交互。首先是显示菜单。然后是根据用户输入的指令，调用对应的函数实现相应的系统功能。具体实现方式是在 while 循环内用一个 switch 语句，循环结束的条件是用户输入的数字是 11（即用户选择“退出”功能）。此外，我还调整了一下输入输出的格式，使得界面看起来更加美观。

代码：

```
#include<iostream>
#include<iomanip>
#include"student.h"
#include"studentData.h"
using namespace std;

extern void title();

int main()
{
    int input = 1;
    string num ;
    studentData WD;
```

```
    cout << endl ;
    cout  <<  "                                                    欢迎进入“学生健康管理系统”" << endl << endl;
    //当输入指令不是 11 时，就默认不退出系统
    while (input != 11)
    {
        cout << "--------------------------------------------------------------------------------" << endl;
        cout  <<  "   1：新建学生健康表  2：添加信息  3：插入信息  4：删除信息  5：修改健康状态" << endl << endl;
        cout << "   6：排序  7：查询  8：输出所有信息  9:读取文件  10：写入文件  11：退出系统" << endl;
        cout << "--------------------------------------------------------------------------------" << endl;
        cin >> input;
        switch (input)
        {
        case 1:
            WD.establish();//建文件
            break;
        case 2:
            WD.add();//添加
            break;
        case 3:
            WD.insert();//插入
            break;
        case 4:
            WD.del();//删除
            break;
        case 5:
            WD.change();//修改
```

```
        break;
    case 6:
        WD.rank();//排序
        break;
    case 7:
        //查询
        cout << "请输入要查找的学号：" << endl;
        cin >> num;
        if (WD.searchNum(num, 0) != 0)
        {
            cout << endl << endl;
            title();
            cout << *WD.searchNum(num, 0) << endl;
        }
        else cout << "未查到相关信息!! ";
        cout << endl << endl;
        break;
    case 8:
        WD.list();//输出所有信息
        break;
    case 9:
        WD.readData();//读取问价
        break;
    case 10:
        WD.writeData();//写入文件
        break;
    case 11:
    {
        cout << "祝您生活愉快，再见！!" << endl;//退出系统
```

```
                break;
            }
            default:
                cout << "This is impossible! Please input correct orders!!!" << endl << endl;
            }
        }
        return 0;
}
```

至此，必做部分结束！

选做部分：

实现思路

在程序启动后，用户可以选择读取已有文件的数据或者新建一个学生健康表，学生对象的数据通过链表的形式临时存储。用户可以根据自身需求，对学生数据进行添加、删除、查找、排序等操作。需要说明的是，用户需要手动点击写入按钮来保存数据，否则，若是通过直接退出程序，是不会实现文件的写入的。

首先对图形化界面进行设计，共设计 11 个界面（菜单界面，新建界面，添加界面，删除界面，修改界面，查找界面，插入界面，排序界面，输出界面，读取界面，写入界面），一个界面实现一个功能。在每一个界面的上方都有所有功能的按钮选项，用户点击相应的按钮，就可以去到相应的操作界面以完成相应操作。

之后，为按钮添加相应的指令，使得用户点击相应的按钮时，切换到相应的操作界面进行相应操作。（如点击添加按钮时，则跳转到添加信息的界面）界面的切换使用 Qt 中的 QStackedLayout 类实现，将相应的界面压入后，按照按钮依次显示。`为了在每一个界面点击相关按钮都可以切换到其他的所有界面，需要对每一个界面的按钮都添加相应的操作。此外，在每一个界面都添加返回菜单按钮和退出按钮，实现返回菜单或退出程序的功能。

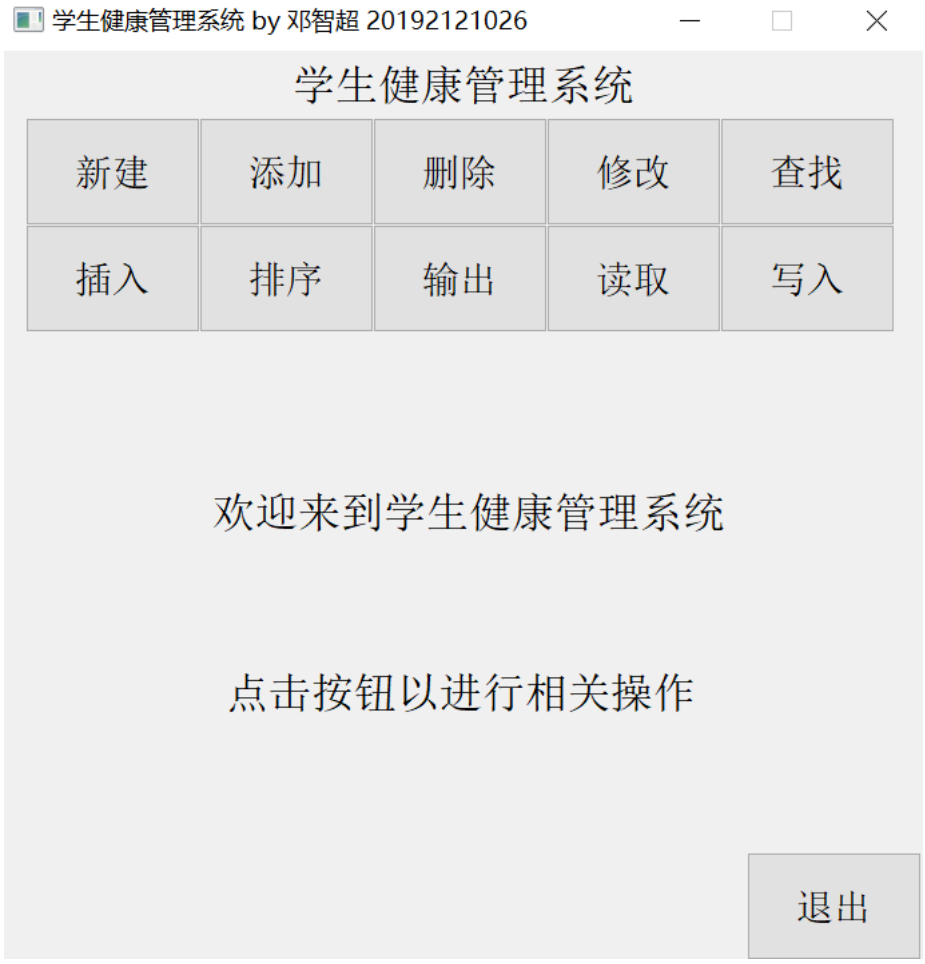
其他相应功能的实现思路：将界面内的各个文本框作为获取用户输入的渠道。根据用户的输入，实例化 student 类对象，对其赋值后用链表连接。之后对链表进

行添加、删除、修改、插入、输出、排序的操作，并将操作的结果反馈输出在相应的文本框之中（如操作成功或失败以及失败的原因等等）。

各个界面的设计视图如下（具体的为按钮添加相关指令稍后会详细说明）。

各个功能的界面视图

菜单界面：



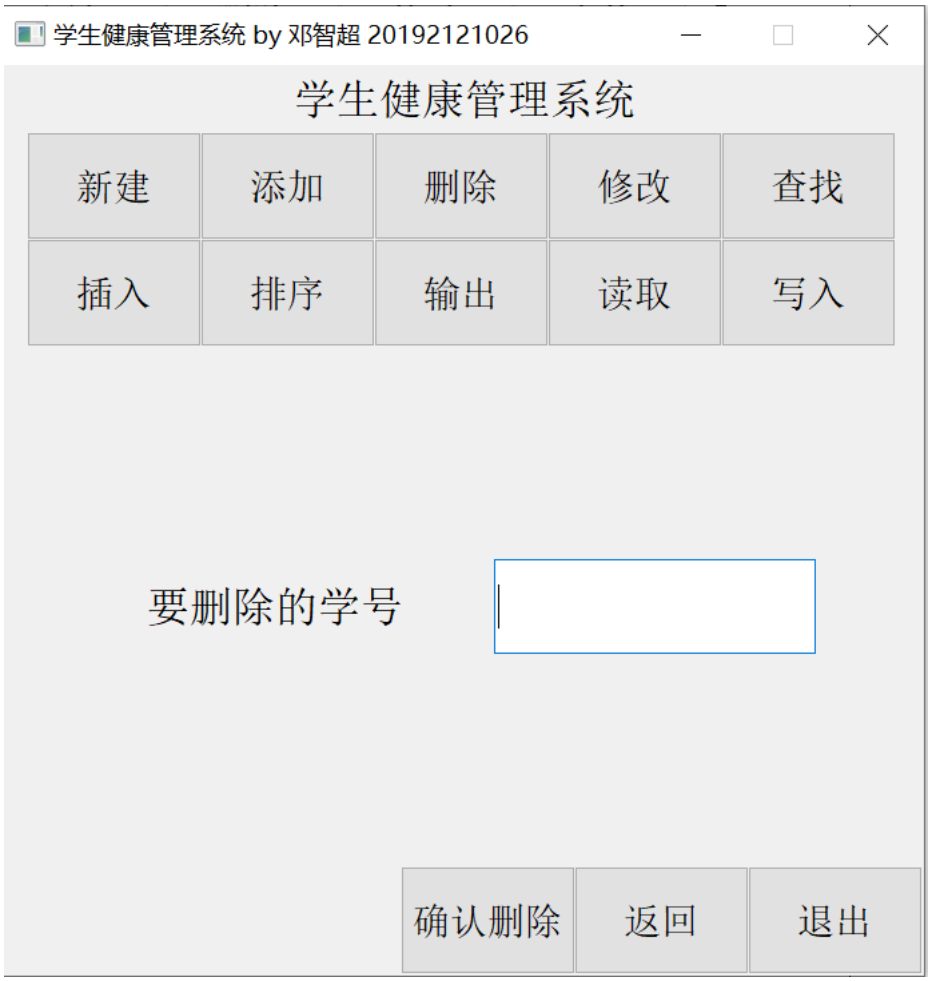
新建界面：


学生健康管理系统 by 邓智超 20192121026
学生健康管理系统
新建
添加
删除
修改
查找
插入
排序
输出
读取
写入
你想建立的表名
确认创建
返回
退出

添加界面：


学生健康管理系统 by 邓智超 20192121026
学生健康管理系统
新建
添加
删除
修改
查找
插入
排序
输出
读取
写入
姓名
性别
学号
出生日期
健康状况
确认添加
返回
退出

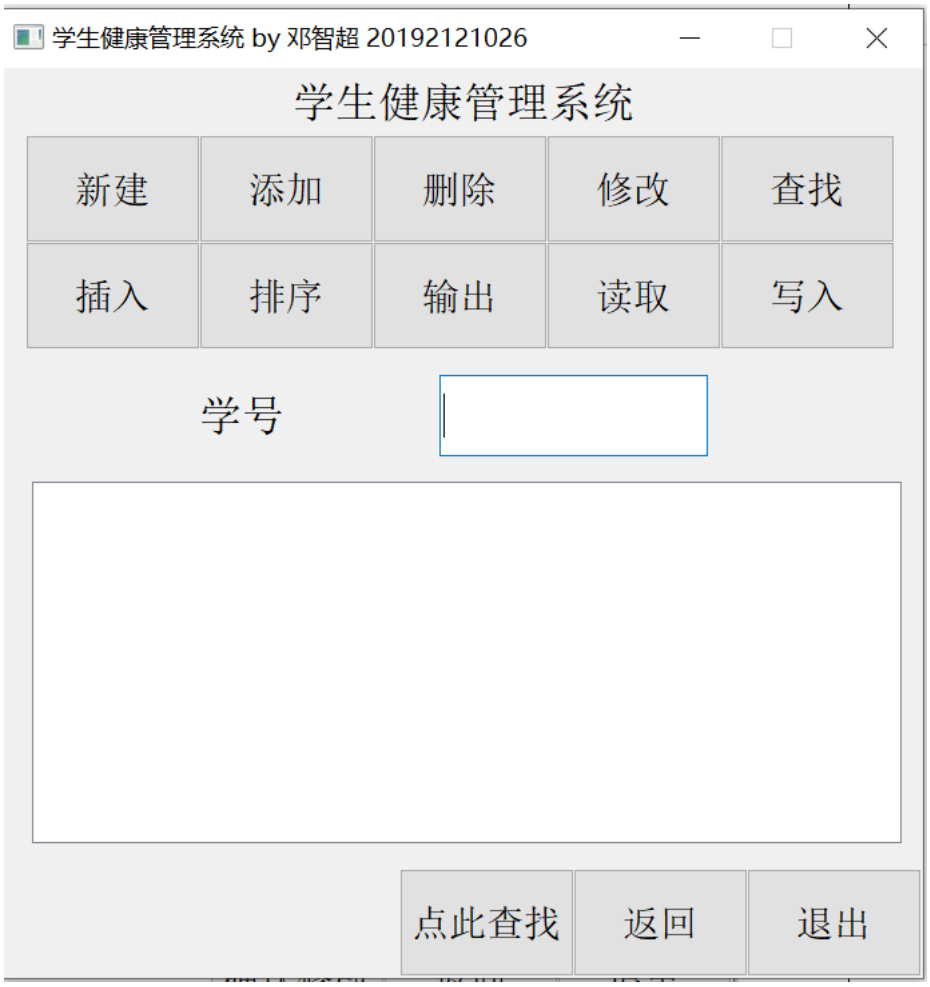
删除界面：


学生健康管理系统 by 邓智超 20192121026
学生健康管理系统
新建
添加
删除
修改
查找
插入
排序
输出
读取
写入
要删除的学号
确认删除
返回
退出

修改界面：


学生健康管理系统 by 邓智超 20192121026
学生健康管理系统
新建
添加
删除
修改
查找
插入
排序
输出
读取
写入
学号
新的健康状况
确认修改
返回
退出

查找界面：


学生健康管理系统 by 邓智超 20192121026
学生健康管理系统
新建
添加
删除
修改
查找
插入
排序
输出
读取
写入
学号
点此查找
返回
退出

插入界面：


学生健康管理系统 by 邓智超 20192121026
学生健康管理系统
新建
添加
删除
修改
查找
插入
排序
输出
读取
写入
位置
性别
姓名
出生日期
学号
健康状况
确认插入
返回
退出

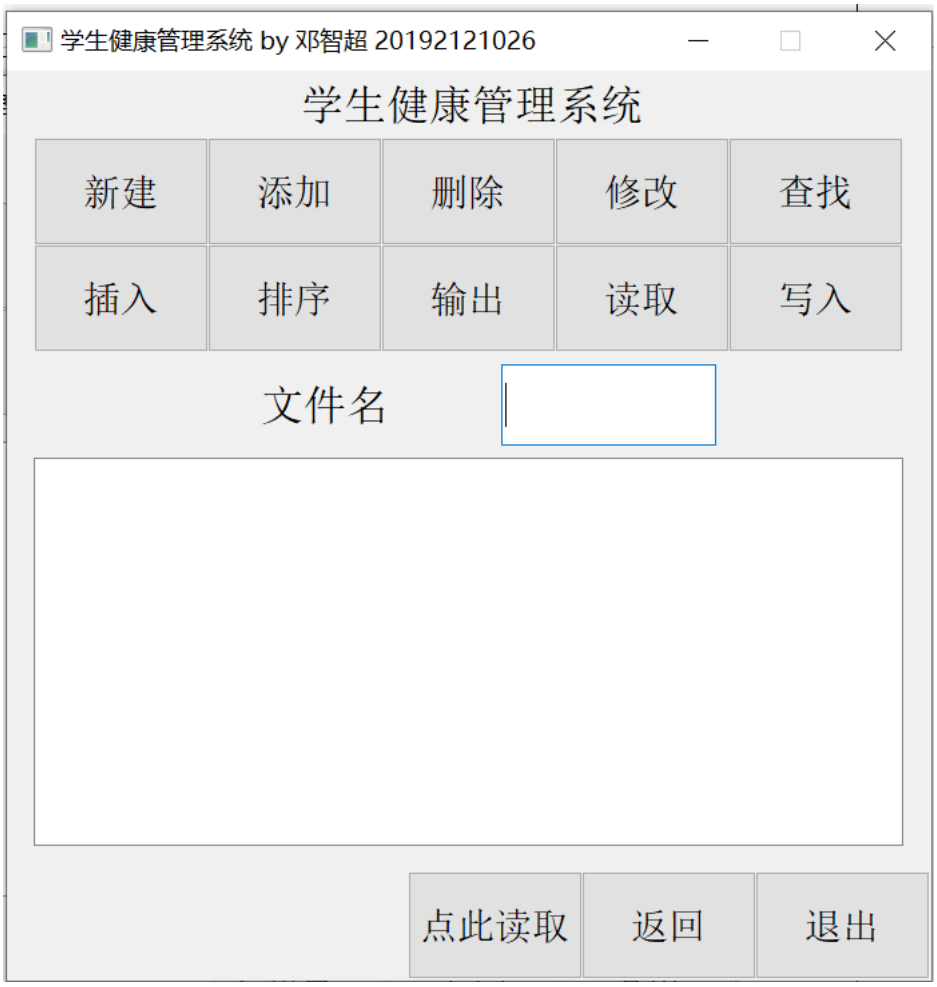
排序界面：


学生健康管理系统 by 邓智超 20192121026
学生健康管理系统
新建
添加
删除
修改
查找
插入
排序
输出
读取
写入
默认从小到大，勾选倒序按钮可按照从大到小排序
倒序
点此排序
返回
退出

输出界面：


学生健康管理系统 by 邓智超 20192121026
学生健康管理系统
新建
添加
删除
修改
查找
插入
排序
输出
读取
写入
点此显示
返回
退出

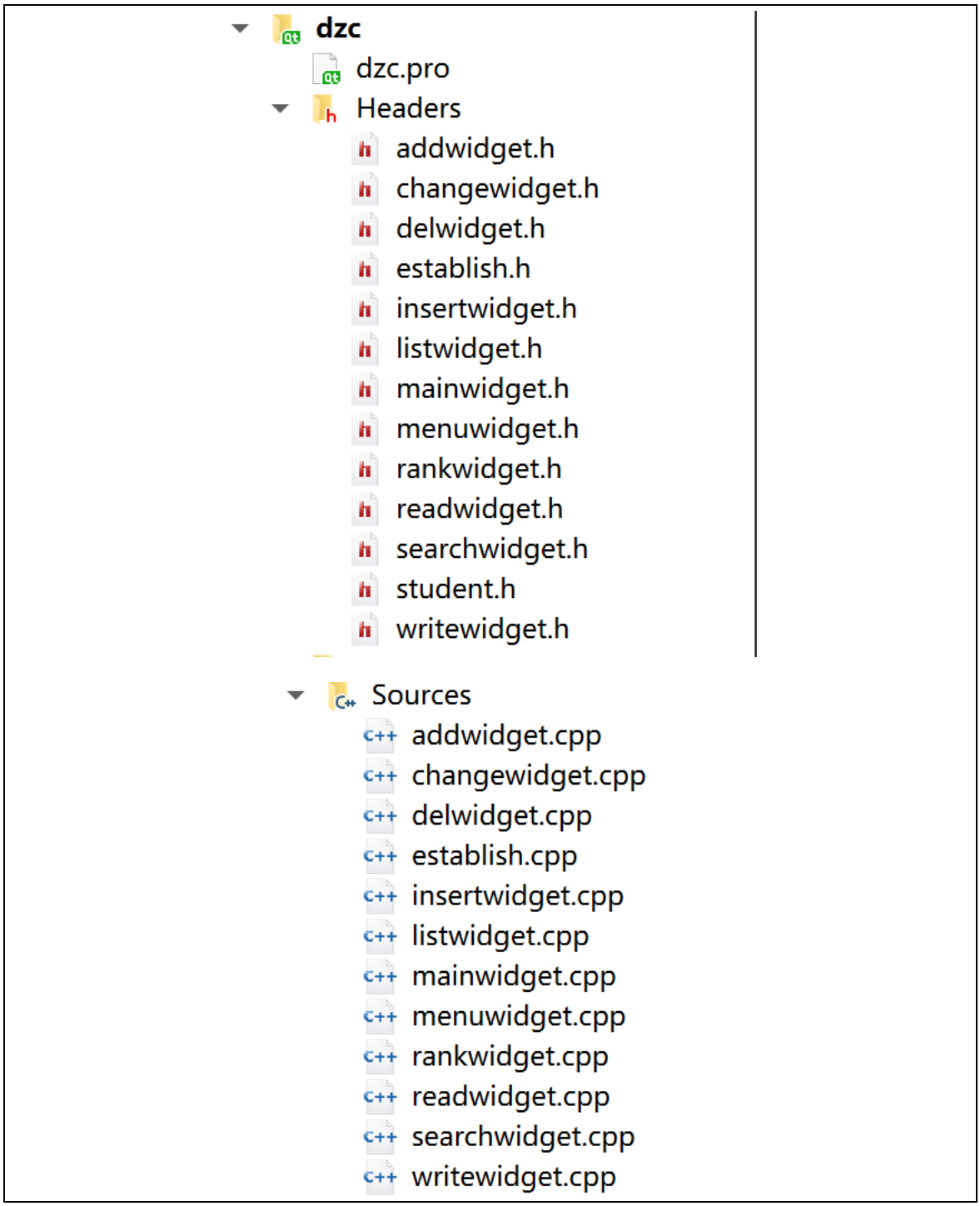
读取界面：

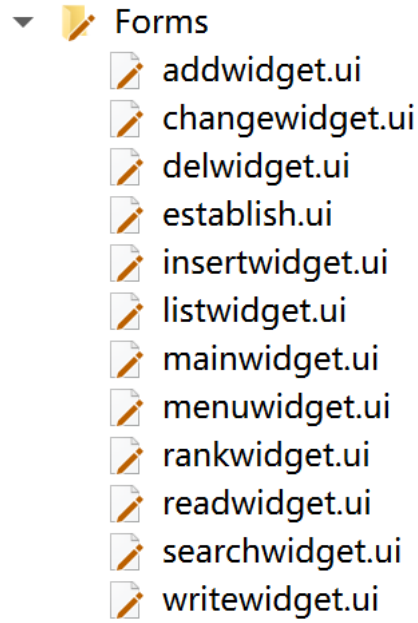

学生健康管理系统 by 邓智超 20192121026
学生健康管理系统
新建
添加
删除
修改
查找
插入
排序
输出
读取
写入
文件名
点此读取
返回
退出

写入界面：

组织架构

添加设计师界面模板 Widget，依次添加菜单窗口 menuWidget，添加窗口 addWidget,删除窗口 delWidget,修改窗口 changeWidget,查找窗口 searchWidget，排序窗口 rankWidget，输出窗口 listWidget，建立窗口 establish，插入窗口 insertWidget，主函数窗口 mainWidget，菜单窗口 menuWidget。之后，添加头文件以实现 student 类的声明的定义。添加完成后，系统结构图如下：


dzc
dzc.pro
Headers
addwidget.h
changewidget.h
delwidget.h
establish.h
insertwidget.h
listwidget.h
mainwidget.h
menuwidget.h
rankwidget.h
readwidget.h
searchwidget.h
student.h
writewidget.h
Sources
addwidget.cpp
changewidget.cpp
delwidget.cpp
establish.cpp
insertwidget.cpp
listwidget.cpp
mainwidget.cpp
menuwidget.cpp
rankwidget.cpp
readwidget.cpp
searchwidget.cpp
writewidget.cpp



之后，按照之前提到的各个界面的设计视图，在各个 ui 文件(除 mainwidget.ui)将系统的视图画出来，并添加上相应的 button（按钮），label（标签），lineEdit（输入文本框），tableWidget（输出框）。为了方便区分，在此约定，将所有界面中用于切换界面的按钮都命名为 addButton(切换到添加界面)，changeButton(切换到修改界面)，delWidget(切换到删除界面)，establishButton（切换到建立界面），insertButton(切换到插入界面)，listButton(切换到输出界面)，rankButton(切换到排序界面)，readButton(切换到读取文件界面)，searchButton(切换到查找界面)，writeButton(切换到写入文件界面)，将所有执行某项操作的按钮都命名为 addingButton(执行添加操作),changingButton（执行修改操作),delingButton(执行删除操作),establishingButton(执行建立操作)，insertingButton(执行插入操作)，listingButton(执行输出操作)，rankingButton(执行排序操作)，readingButton(执行读取操作)，searchingButton(执行查找操作)，writingButton(执行写入文件操作)。返回菜单按钮命名为 returnButton，退出程序按钮命名为 exitButton。

实现界面切换

使用头文件 QStackedLayout 中的 QStackedLayout 类实现窗口切换，在 mainWidget 中实现，main 函数写在 mainWidget.cpp 中。

在 mainWidget.h 中修改 mainWidget 的定义，为其添加私有成员（添加指向

addWidget 类,changeWidget 类,delWidget 类,establishWidget 类,insertWidget 类,listWidget 类,rankWidget 类,readWdiget 类,searchWidget 类,writeWidget 类的指针）。代码如下：

```
#ifndef MAINWIDGET_H
#define MAINWIDGET_H
#include <QWidget>
#include<QStackedLayout>
#include"addwidget.h"
#include"changewidget.h"
#include"delwidget.h"
#include"establish.h"
#include"insertwidget.h"
#include"listwidget.h"
#include"rankwidget.h"
#include"readwidget.h"
#include"searchwidget.h"
#include"student.h"
#include"writewidget.h"
#include"menuwidget.h"
namespace Ui {
class MainWidget;
}
class MainWidget : public QWidget
{
    Q_OBJECT
public:
    explicit MainWidget(QWidget *parent = nullptr);
    ~MainWidget();
private:
```

```
        Ui::MainWidget *ui;
        QStackedLayout *stackLayout;
        addWidget *addwidget;
        changeWidget *changewidget;
        delWidget *delwidget;
        establish* establishwidget;
        insertWidget*insertwidget;
        listWidget*listwidget;
        MenuWidget*menuwidget;
        rankWidget*rankwidget;
        readWidget*readwidget;
        searchWidget*searchwidget;
        writeWidget*writewidget;
    };
    #endif // MAINWIDGET_H
```

之后在 mainWidget.cpp 中将 settingWidget 设为 stackLayout 布局，在构造函数中显式实例化每一个需要的窗口，并将他们加入 stackeLayout 布局中去。在构造函数中，顺便将窗口名称设置成“学生健康管理系统 by 邓智超 20192121026”，窗口大小设置成 692*682。

代码：

```
MainWidget::MainWidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::MainWidget)
{
    setWindowTitle("学生健康管理系统 by 邓智超 20192121026");
    setFixedSize(692,682);
    ui->setupUi(this);
```

```
addwidget=new addWidget;
changewidget=new changeWidget;
delwidget=new delWidget;
establishwidget=new establish;
insertwidget=new insertWidget;
listwidget=new listWidget;
menuwidget=new MenuWidget;
rankwidget=new rankWidget;
readwidget=new readWidget;
searchwidget=new searchWidget;
writewidget=new writeWidget;
stackLayout=new QStackedLayout;
stackLayout->addWidget(menuwidget);
stackLayout->addWidget(addwidget);
stackLayout->addWidget(changewidget);
stackLayout->addWidget(insertwidget);
stackLayout->addWidget(delwidget);
stackLayout->addWidget(readwidget);
stackLayout->addWidget(writewidget);
stackLayout->addWidget(rankwidget);
stackLayout->addWidget(searchwidget);
stackLayout->addWidget(listwidget);
stackLayout->addWidget(establishwidget);

setLayout(stackLayout);
```

然后，使用 connect 函数，将信号 display 和槽函数连接起来代码：（display 是信号，之后会提到），这样调用 display 函数时，就可以显示不同的界面了。代码如下：

```
connect(menuwidget,&MenuWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(addwidget,&addWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(changewidget,&changeWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(insertwidget,&insertWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(delwidget,&delWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(readwidget,&readWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(writewidget,&writeWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(rankwidget,&rankWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(searchwidget,&searchWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(listwidget,&listWidget::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
connect(establishwidget,&establish::display,stackLayout,
        &QStackedLayout::setCurrentIndex);
```

按照以上的 stackLayout 布局，menu 界面，add 界面，change 界面，insert 界面，del 界面，read 界面，write 界面，rank 界面，search 界面，list 界面，establish 界面的序号分别是 0，1，2，3，4，5，6，7，8，9，10（这个编号会在之后的信号函数 display 中用到）。之后，在每个窗口的头文件中都添加 signals，并在窗口的 cpp 文件中将相应的按钮和对应的信号匹配好，就可以切换不同的界面

了。**接下来以 memu 窗口的实现文件为例：**

首先在 menuWidget.h 中添加槽函数和信号，其中每一个函数对应点击某按钮后执行的界面切换的动作（之前已经将信号和槽函数的连接实现）。

```
private slots:
    void on_addButton_clicked();
    void on_changeButton_clicked();
    void on_delButton_clicked();
    void on_establishButton_clicked();
    void on_insertButton_clicked();
    void on_listButton_clicked();
    void on_rankButton_clicked();
    void on_readButton_clicked();
    void on_searchButton_clicked();
    void on_writeButton_clicked();
    void on_exitButton_clicked();
signals:
        void display(int number);
```

之后，在 menuWidget.cpp 中给出各个函数的实现：

```
void MenuWidget::on_addButton_clicked()
{
    emit display(1);
}
void MenuWidget::on_changeButton_clicked()
{
    emit display(2);
}
void MenuWidget::on_insertButton_clicked()
{
```

```
    emit display(3);
}
void MenuWidget::on_delButton_clicked()
{
    emit display(4);
}
void MenuWidget::on_readButton_clicked()
{
    emit display(5);
}
void MenuWidget::on_writeButton_clicked()
{
    emit display(6);
}
void MenuWidget::on_rankButton_clicked()
{
    emit display(7);
}
void MenuWidget::on_searchButton_clicked()
{
    emit display(8);
}
void MenuWidget::on_listButton_clicked()
{
    emit display(9);
}
void MenuWidget::on_establishButton_clicked()
{
    emit display(10);
```

```
}
void MenuWidget::on_exitButton_clicked()
{
    QApplication::exit();
}
```

至此，就实现了每个界面之间的切换，用户在任意界面点击任意的按钮，都可以切换到任意操作界面。接下来按照各个窗口介绍实现方式。

Student 类的声明与实现

因为 student 类的实现方式较简单,所以这里将类的声明和实现都放在头文件中了。Student 类的私有成员用于存储学生对象的数据，包括 name(姓名)，number(学号)，gender(性别)，dateOfBirth(出生日期)和 healthCondition(健康状况)，这些属性都声明为 QString 类。公有成员有：用于指向链表下一节点的指针 student*next、设置属性函数(作用是对学生的属性进行设置)setName()、setNumber()、setGender()、setBirthday()、getHealthCondition()，以及一堆返回信息的函数(getName()、getBirthday()、getGender()、getHealthCondition()和 getNumber，实现方法很简单，对于设置属性函数，将对象的属性设置成相应的值即可，对于返回信息的函数，返回对象的对应属性即可，在此不做详细说明)、exchange()函数（交换两个对象的数据，在选择排序的时候便于交换两个对象的数据，具体的实现方法就是利用中间变量 temp，将两个对象的每个属性都交换即可）。

声明和实现 Student 类的代码如下：

```
#ifndef STUDENT_H
#define STUDENT_H
#include<QString>
using namespace std;
class student
{
private:
```

```
    QString name;
    QString number;
    QString gender;
    QString birthday;
    QString healthCondition;
public:
    student * next;
    QString getName(){return name;}
    QString getNumber(){return number;}
    QString getGender(){return gender;}
    QString getBirthday(){return birthday;}
    QString getHealthCondition(){return healthCondition;}
    void setName(QString n){name=n;}
    void setNumber(QString n){number=n;}
    void setGender(QString n){gender=n;}
    void setBirthday(QString n){birthday=n;}
    void setHealthCondition(QString n){healthCondition=n;}
    void exchange(student& w2)
    {
        QString temp1;
        temp1 = this->name;//交换姓名
        this->name = w2.name;
        w2.name = temp1;
        temp1 = this->number;//交换学号
        this->number = w2.number;
        w2.number = temp1;
        temp1 = this->gender;//交换性别
        this->gender = w2.gender;
        w2.gender = temp1;
```

```
        temp1 = this->birthday;//交换出生日期
        this->birthday = w2.birthday;
        w2.birthday = temp1;
        temp1 = this->healthCondition;//交换健康状况
        this->healthCondition = w2.healthCondition;
        w2.healthCondition = temp1;
    }
};
#endif // STUDENT_H
```

addWidget 实现

此模块的功能是实现添加操作。首先在 addWidget.ui 界面将用于接收姓名，学号，性别，出生日期，健康状况的文本框分别命名为 nameLineEdit，numberLineEdit,genderLineEdit,birthdayLineEdit,healthConditionLineEdit，便于在编写代码时对其进行区分。

之后在 addWidget.cpp 中声明全局变量：①指向 student 类的指针 first，并赋初为 0 ②QString 类对象 fileName（用于存储文件名）并赋初值为 0 ③整型变量 sum，并赋初值为 0 这些变量有的在此用不到，但是在整个系统中，需要使用到。所以就在此一并声明了。接下来说明 addingButton 被点击后的执行操作。

函数原型：

void addWidget::on_addingButton_clicked()

实现方式：

首先声明 QString 类对象 name，number，gender，birthday，healthCondition，将用户的输入分别从 nameLineEdit， numberLineEdit， genderLineEdit，birthdayLineEdit，healthConditionLineEdit 读出，并一一赋值给上述的五个变量。之后，检查用户是否已经输入五个所需数据，若没有输入完整，则输出提示信息，清除用户输入后结束函数。若输入完整，则判断用户信息（通过学号判断）是否重复，若重复，就输出提示信息并退出程序。开辟一个 student 类的对象空间，将之前从用户获取的输入赋值给 student 对象，然后采用头插入的方式接入链表。并将数据

总数 sum 加 1。

代码：

```
void addWidget::on_addingButton_clicked()//点击 addingButton 后进行的操作
{
    //将用户输入的姓名、学号、性别、生日、健康状况由文本框取出
    QString name = ui->nameLineEdit->text();
    QString number = ui->numberLineEdit->text();
    QString gender = ui->genderLineEdit->text();
    QString birthday = ui->birthdayLineEdit->text();
    QString healthCondition = ui->healthConditionLineEdit->text();
    //判断用户的输入数据是否完整
    if(ui->nameLineEdit->text() == "" || ui->numberLineEdit->text() == ""
            || ui->genderLineEdit->text() == "" || ui->birthdayLineEdit->text() == ""
            || ui->healthConditionLineEdit->text() == "")
    {
        //输出提示信息
        QMessageBox::about(NULL, "output", "please fill all the blank");
        return;
    }
    //对链表遍历，判断信息是否重复
    student *current=first;
    while(first!=0&&current->getNumber()!=number)
    {
        current=current->next;
        if(current==0)break;
    }
    //若重复，输出提示信息并清空用户输入
    if(current!=0)
        {
```

```
        QMessageBox::about(NULL, "output", "information repeated");
        ui->nameLineEdit->clear();
        ui->numberLineEdit->clear();
        ui->genderLineEdit->clear();
        ui->birthdayLineEdit->clear();
        ui->healthConditionLineEdit->clear();
        return;
    }
    //若不重复，执行插入操作
    else
    {
        student *p=new student;
        p->next=first;
        first=p;
        p->setName(name);
        p->setNumber(number);
        p->setGender(gender);
        p->setBirthday(birthday);
        p->setHealthCondition(healthCondition);
        //输出提示信息
        QMessageBox::about(NULL, "output", "seccess");
        //总数加 1
        sum++;
    }
    //清空文本框输入
    ui->nameLineEdit->clear();
    ui->numberLineEdit->clear();
    ui->genderLineEdit->clear();
    ui->birthdayLineEdit->clear();
```

```
    ui->healthConditionLineEdit->clear();
}
```

之后的各窗口的 ui 界面设置都是类似的，接下来的模块中将不在介绍， 仅仅介绍点击按钮后执行的操作。

changeWidget 实现

函数原型：

void changeWidget::on_changingButton_clicked()

实现方式：

此模块的功能是修改学生的健康状况。首先检查用户是否已经输入两个所需数据，若没有输入完整，则输出提示信息，清除用户的输入后结束函数。之后声明 QString 类对象 number，healthCondition，将用户的输入分别从 numberLineEdit，healthConditionLineEdit 读出，并赋值给上述变量。之后，判断链表是否为空，若为空，输出提示并结束。若不为空，则遍历链表判断用户信息（通过学号判断）是否存在目标信息，若不存在，就输出提示信息并退出程序。若存在，则将目标节点的健康状况改成新输入的健康状况。

代码：

```
void changeWidget::on_changingButton_clicked()
{
    //判断用户是否输入完整
    if(ui->numberLineEdit->text() == ""|| ui->healthConditionLineEdit->text() == "")
    {
        //输出提示信息
        QMessageBox::about(NULL, "output", "please fill all the blank");
        //清空用户输入
        ui->numberLineEdit->clear();
        ui->healthConditionLineEdit->clear();
        return;
```

```
}
//将用户输入的学号和新的健康状况由输入文本框取出
QString number = ui->numberLineEdit->text();
QString healthCondition = ui->healthConditionLineEdit->text();
student *current=first;
//若链表为空，输出提示
if(first==0)
{
    QMessageBox::about(NULL, "output", "no information");
    ui->numberLineEdit->clear();
    ui->healthConditionLineEdit->clear();
    return;
}
//判断目标节点是否存在
while(current->getNumber()!=number)
{
    current=current->next;
    if(current==0)break;
}
//若不存在，输出提示
if(current==0)
{
    QMessageBox::about(NULL, "output", "no information");
    ui->numberLineEdit->clear();
    ui->healthConditionLineEdit->clear();
    return;
}
//若存在，执行修改操作
else
```

```
    {
        current->setHealthCondition(healthCondition);
        QMessageBox::about(NULL, "output", "success");
        //清空输入
        ui->numberLineEdit->clear();
        ui->healthConditionLineEdit->clear();
    }
}
```

insertWidget 实现

函数原型：

```
void insertWidget::on_insertingButton_clicked()
```

实现方式：

此模块的功能是实现在指定位置插入学生信息。首先声明 QString 类对象 name，number，gender，birthday，healthCondition 和整型对象 address，将用户的输入分别从 nameLineEdit， numberLineEdit， genderLineEdit，birthdayLineEdit，healthConditionLineEdit，addressLineEdit 读出，并一一赋值给上述的变量。之后，检查用户是否已经输入 6 个所需数据，若没有输入完整，则输出提示信息，清除用户输入后结束函数。若输入完整，则判断用户信息（通过学号判断）是否重复，若重复，就输出提示信息并退出程序。若不重复，则将指针移动到待插入位置，若插入位置是 1，则直接使用头插入方式插入即可，若插入位置大于链表长度，则输出提示并默认插入到最后位置，其他所有情况，插入后将前后节点都和插入节点连接即可。最后将数据总数 sum 加 1。

代码：

```
void insertWidget::on_insertingButton_clicked()
{
    //将用户输入由文本框取出
    QString number = ui->numberLineEdit->text();
```

```
    QString name = ui->nameLineEdit->text();
    QString gender = ui->genderLineEdit->text();
    QString birthday = ui->birthdayLineEdit->text();
    QString healthCondition = ui->healthConditionLineEdit->text();
    if(ui->addressLineEdit->text()==""||ui->nameLineEdit->text() == "" ||
ui->numberLineEdit->text() == ""
            || ui->genderLineEdit->text() == "" || ui->birthdayLineEdit->text() == ""
            || ui->healthConditionLineEdit->text() == "")
    {
        //输出提示信息
        QMessageBox::about(NULL, "output", "please fill all the blank");
        return;
    }
    //判断是否重复
    student *current=first;
    while(current->getNumber()!=number)
    {
        current=current->next;
        if(current==0)break;
    }
    //若重复，输出提示
    if(current!=0)
    {
        QMessageBox::about(NULL, "output", "information repeated");
        ui->numberLineEdit->clear();
        return;
    }
    //若链表为空，直接插入
    if(first==0)
```

```
{
    student*p=new student;
    //赋值
    p->setName(name);
    p->setNumber(number);
    p->setGender(gender);
    p->setBirthday(birthday);
    p->setHealthCondition(healthCondition);
    first=p;
    first->next=0;
}
//将用户输入的位置由取出
int address = ui->addressLineEdit->text().toInt();
current=first;
//将指针移动到待插入位置
for (int i = 1; i <= address-2; i++)
{
    current = current->next;
    if (current->next == 0&&i!=address-2)
    {
        //若链表长度小于用户输入
        QMessageBox::about(NULL, "output", "length error,it'll be added at last address");
        break;
    }
}
//执行插入操作
student* p = new student;
//赋值
p->setName(name);
```

```
    p->setNumber(number);
    p->setGender(gender);
    p->setBirthday(birthday);
    p->setHealthCondition(healthCondition);
    //直接插入头节点
    if(address==1)
    {
        p->next=first;
        first=p;
    }
    //节点前后连接
    else
    {
        p->next = current->next;
        current->next = p;
    }
    QMessageBox::about(NULL,"output","success");
    //清空输入
    ui->nameLineEdit->clear();
    ui->numberLineEdit->clear();
    ui->genderLineEdit->clear();
    ui->birthdayLineEdit->clear();
    ui->healthConditionLineEdit->clear();
    ui->addressLineEdit->clear();
    sum++;
}
```

delWidget 实现

函数原型：

void delWidget::on_delingButton_clicked()

实现方式：

此模块的功能是实现删除某学生的信息。获取用户输入的数据后，对链表进行遍历（按照学号查找），若无目标节点，则输出提示信息并结束函数，若目标节点存在，若目标节点是 first 节点，则利用中间变量将 first 指向的内存空间释放，并将 first 赋值为 first->next。若目标节点不是第一个节点，则再次遍历链表找到目标节点的前一个节点，将目标节点的前一节点和后一节点连接，然后释放目标节点的空间。最后将数据总数 sum 减 1。

代码：

```
void delWidget::on_delingButton_clicked()
{
    //将用户输入的学号由 numberLineEdit 取出
    QString number = ui->numberLineEdit->text();
    student *current=first;
    //链表为空，则输出提示
    if(first==0)
    {
        QMessageBox::about(NULL, "output", "no information");
        ui->numberLineEdit->clear();
        return;
    }
    //判断有无目标信息
    while(current->getNumber()!=number)
    {
        current=current->next;
        if(current==0)break;
    }
    //若无，输出提示
    if(current==0)
```

```
        {
            QMessageBox::about(NULL, "output", "no information");
            ui->numberLineEdit->clear();
            return;
        }
        //若有，执行删除操作
        else
        {
            //头节点为目标节点
            if(current==first)
            {
                first=first->next;
                delete current;
            }
            //目标节点不是头节点
            else
            {
                student *prenode=first;
                //获取目标节点的前一节点
                while((prenode->next)->getNumber()!=number)
                    prenode=prenode->next;
                prenode->next=current->next;
                delete current;
            }
             QMessageBox::about(NULL, "output","success!");
             //总数减 1
             sum--;
        }
}
```

readWidget 实现

函数原型：

```
void readWidget::on_readingButton_clicked()
```

实现方式：

此模块的功能是将文件中的学生信息读取到系统中用链表连接后全部输出。获取用户输入的文件名，在其输入名字后加上“.txt”，作为打开的文件名。若此文件为空或者不存在，则输出提示信息并结束函数。若不为空，则利用文件流，一次读取 name，number，gender，birthday，healthCondition 这五个数据将这五个数据赋值给 student 类的对象，并将此对象采用头插入的方式插入链表。读取完成后，因为文件会多读取一行，所以需要删去最前面的一个节点。然后对链表进行遍历，将学生数据输出在 tableWidget 中。首先设置输出列各列的名字为姓名、学号、性别、出生日期、 健康状况，之后将对象的数据输出在对应的行和列。

代码：

```
void readWidget::on_readingButton_clicked()
{
    //将用户输入的文件由 nameLineEdit 取出
    QString name = ui->nameLineEdit->text();
    name=name+".txt";
    QFile file(name);
    file.open(QIODevice::ReadOnly|QIODevice::Text);
    //以只读方式打开文本文件
    QString number;
    QString gender;
    QString birthday;
    QString healthCondition;
    if(!file.isOpen())
    {
```

```
        //文件打开失败
        QMessageBox::about(NULL, "output", "fail to open the file");
        return ;
    }
    student*p;
    QTextStream in(&file);
    while(!in.atEnd())
    {
        //读到文件结尾
        in>> name >> number >> gender >> birthday >> healthCondition;
        p=new student;
        p->setName(name);
        p->setNumber(number);
        p->setGender(gender);
        p->setBirthday(birthday);
        p->setHealthCondition(healthCondition);
        p->next = first;//头插入方式，可以避免判断链表是否为空
        first = p;
        sum++;
    }
    file.close();
    //因为文件会多读取一行，所以将最后一次读取的内容删除
    student *q=first;
    first=first->next;
    delete q;
    sum--;
    //若链表为空，输出提示
    if (first == 0)
    {
```

```
            QMessageBox::about(NULL, "output", "error:no information in the txt");
            return;
        }
        QMessageBox::about(NULL, "output", "success");
        student *current=first;
        int i=0;
        while(current!=0)
        {
            //设置每行五列
            ui->tableWidget->setColumnCount(5);
            QStringList headerLabels;
            //设置列名称
            headerLabels << tr("姓名") << tr("学号") << tr("性别") << tr("出生日期") << tr("健康状
况") ;
            ui->tableWidget->setHorizontalHeaderLabels(headerLabels);
            ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
            //设置行数为数据总数
            ui->tableWidget->setRowCount(sum);//sum 行数据
            //输出
            ui->tableWidget->setItem(i, 0, new QTableWidgetItem(current->getName()));
            ui->tableWidget->setItem(i, 1, new QTableWidgetItem(current->getNumber()));
            ui->tableWidget->setItem(i, 2, new QTableWidgetItem(current->getGender()));
            ui->tableWidget->setItem(i, 3, new QTableWidgetItem(current->getBirthday()));
            ui->tableWidget->setItem(i++, 4, new
QTableWidgetItem(current->getHealthCondition()));
            current=current->next;
        }
}
```

writeWidget 实现

函数原型：

```
void writeWidget::on_writingButton_clicked()
```

实现方式：

此模块的功能是实现将链表中的学生数据写入到文件中。用户之前已经建立了健康表（即全局变量 fileName 不为空），则默认将系统数据存储在以之前输入的健康表名字为文件名的文件中（文件名存储在 fileName 变量）中，若 fileName 为空（即之前没有建立健康表），则提示用户输入文件名。以只写的方式打开文件后，对链表进行遍历，利用文件流将学生数据依次写入文件。

代码：

```
void writeWidget::on_writingButton_clicked()
{
    //若之前未输入文件名
    if(fileName=="")
    {
        fileName=ui->nameLineEdit->text();
        if(fileName=="")
        QMessageBox::about(NULL,"output","请输入文件名");
        return;
    }
    QFile file(fileName+".txt");
    //以只写的方式打开
    file.open(QIODevice::WriteOnly|QIODevice::Text);
    if(!file.isOpen())
    {
        //如果数据文件没有打开，弹出对话框提示用户
        QMessageBox::about(NULL, "output", "fail to open the txt");
```

```
        return;
    }
    //设置文件流
    QTextStream out(&file);
    student *current=first;
    //遍历链表，写入数据
    while(current!=0)
    {
        out <<current->getName()<< " " <<current->getNumber() <<
            " " <<current->getGender()<< " " << current->getBirthday()
            << " " << current->getHealthCondition()<<" ";
        current=current->next;
    }
    file.close();
    QMessageBox::about(NULL, "output", "success");
}
```

rank 实现

函数原型：

```
void rankWidget::on_rankingButton_clicked()
```

实现方式：

此模块的功能是实现将学生信息进行正序或者倒序排列，并输出排列结果。首先判断链表是否为空，若为空，则输出提示信息并退出函数。若不为空，就判断 checkButton 有没有被勾选，若勾选 checkButton，则对链表执行从大到小的选择排序，若未勾选 checkButton，则进行从小到大的选择排序。排序时，通过调用之前声明的 exchange 函数交换两个对象的数据。排序完成后，对链表进行遍历，将学生数据输出在 tableWidget 中。首先设置输出列各列的名字为姓名、学号、性别、出生日期、 健康状况，之后将对象的数据输出在对应的行和列。

代码：

```
void rankWidget::on_rankingButton_clicked()
{
    //判断链表是否为空
    if (first == 0)
    {
        QMessageBox::about(NULL, "反馈", "ERROR：信息库为空，无法排序，请输入数据！
");
        return ;
    }
    student* p=0;
    student* q=0;
    //选择排序
    for(p=first;p!=0;p=p->next)
        for (q = p->next; q != 0; q = q->next)
        {
            //若勾选倒叙按钮
            if(ui->checkBox->isChecked())
            {
                if (q->getNumber() > p->getNumber())
                    p->exchange(*q);//交换数据
            }
            //若未勾选
            else
            {
                if (q->getNumber() < p->getNumber())
                    p->exchange(*q);//交换数据
            }
        }
```

```
    QMessageBox::about(NULL, "反馈", "排序成功");
    student *current=first;
    int i=0;
    //输出链表内容
    while(current!=0)
    {
        //设置列数为 5
        ui->tableWidget->setColumnCount(5);
        QStringList headerLabels;
        //设置列名称
        headerLabels << tr("姓名") << tr("学号") << tr("性别") << tr("出生日期") << tr("健康") ;
        ui->tableWidget->setHorizontalHeaderLabels(headerLabels);
        ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
        //设置行数为 sum
        ui->tableWidget->setRowCount(sum);
        //输出
        ui->tableWidget->setItem(i, 0, new QTableWidgetItem(current->getName()));
        ui->tableWidget->setItem(i, 1, new QTableWidgetItem(current->getNumber()));
        ui->tableWidget->setItem(i, 2, new QTableWidgetItem(current->getGender()));
        ui->tableWidget->setItem(i, 3, new QTableWidgetItem(current->getBirthday()));
        ui->tableWidget->setItem(i++,4, new
QTableWidgetItem(current->getHealthCondition()));
        current=current->next;
    }
}
```

search 实现

函数原型：

```
    void searchWidget::on_searchingButton_clicked()
```

实现方式：

此模块的功能是按照学号查找学生信息并将学生信息输出。获取用户输入后，对链表进行遍历，判断目标节点（按照学号查找）在不在链表中，若不在，则输出提示信息并结束函数，若在链表中，则将目标节点信息输出在 tableWidget 中，首先设置输出列各列的名字为姓名、学号、性别、出生日期、 健康状况，设置列数为 5，行数为 1，之后将目标节点的数据输出在对应的行。

代码：

```
void searchWidget::on_searchingButton_clicked()
{
    //将用户输入的学号由 numberLineEdit 取出
    QString number = ui->numberLineEdit->text();
    student *current=first;
    //若链表为空。输出提示
    if(first==0)
    {
        QMessageBox::about(NULL, "output", "no information");
        ui->numberLineEdit->clear();
        return;
    }
    //判断有无目标节点
    while(current->getNumber()!=number)
    {
        current=current->next;
        if(current==0)break;
    }
    //若无，输出提示
    if(current==0)
    {
        QMessageBox::about(NULL, "output", "no information");
```

```
            ui->numberLineEdit->clear();
            return;
        }
        else
        {
            //设置列数为 5
            ui->tableWidget->setColumnCount(5);
            QStringList headerLabels;
            //设置列名称
            headerLabels << tr("姓名") << tr("学号") << tr("性别") << tr("出生日期") << tr("健康") ;
            ui->tableWidget->setHorizontalHeaderLabels(headerLabels);
            ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
            //设置行数为 1
            ui->tableWidget->setRowCount(1);
            //输出
            ui->tableWidget->setItem(0, 0, new QTableWidgetItem(current->getName()));
            ui->tableWidget->setItem(0, 1, new QTableWidgetItem(current->getNumber()));
            ui->tableWidget->setItem(0, 2, new QTableWidgetItem(current->getGender()));
            ui->tableWidget->setItem(0, 3, new QTableWidgetItem(current->getBirthday()));
            ui->tableWidget->setItem(0, 4, new QTableWidgetItem(current->getHealthCondition()));
            ui->numberLineEdit->clear();
        }
}
```

list 实现

函数原型：

```
void listWidget::on_listingButton_clicked()
```

实现方式：

此模块的功能是实现将链表中的学生信息全部输出。首先判断链表是否为空，若为空，则输出提示信息并结束函数。若不为空，则对链表进行遍历，将学生数据输出在 tableWidget 中。首先设置输出列各列的名字为姓名、学号、性别、出生日期、 健康状况，设置列数为 5，行数为 sum，之后将对象的数据输出在对应的行和列。

代码：

```
void listWidget::on_listingButton_clicked()
{
    //若链表为空
    if(first==0)
        QMessageBox::about(NULL, "反馈", "信息为空");
    //若不为空
    else
    {
        student *current=first;
        int i=0;
        while(current!=0)
        {
            //设置列数为 5
            ui->tableWidget->setColumnCount(5);
            QStringList headerLabels;
            //设置列名称
            headerLabels << tr("姓名") << tr("学号") << tr("性别") << tr("出生日期") << tr("健康");
            ui->tableWidget->setHorizontalHeaderLabels(headerLabels);

ui->tableWidget->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);
            //设置行数为 sum
            ui->tableWidget->setRowCount(sum);
```

```
            //输出
            ui->tableWidget->setItem(i, 0, new QTableWidgetItem(current->getName()));
            ui->tableWidget->setItem(i, 1, new QTableWidgetItem(current->getNumber()));
            ui->tableWidget->setItem(i, 2, new QTableWidgetItem(current->getGender()));
            ui->tableWidget->setItem(i, 3, new QTableWidgetItem(current->getBirthday()));
            ui->tableWidget->setItem(i++, 4, new
QTableWidgetItem(current->getHealthCondition()));
            current=current->next;
        }
    }
}
```

establish 实现

函数原型：

```
void establish::on_establishingButton_clicked()
```

实现方式：

获取用户输入并赋值给 fileName 即可。功能说明：此处的 fileName 将作为之后写入文件的文件名，若用户已经建立健康表（即 fileName 不为空），则写入文件时，直接默认以此名为文件名，若未输入健康表名（即 fileName 为空），则写入文件时，会提示用户输入文件名。

代码：

```
void establish::on_establishingButton_clicked()
{
    fileName = ui->nameLineEdit->text();
    //将用户输入的姓名由 nameLineEdit 取出
    if(fileName=="")
    {
        QMessageBox::about(NULL, "output", "please fill all the blank");
```

```
    }
    else
    {
        QMessageBox::about(NULL, "output", "success");
        ui->nameLineEdit->clear();
    }
}
```

Main 函数

实现方式：

声明一个 QApplication 类的对象 app 和 mainWidget 类对象 m，调用 m.show()函数显示菜单即可。然后用户就可以使用该系统了。

代码：

```
int main(int argc,char*argv[])
 {
     QApplication app(argc,argv);
     MainWidget m;
     m.setWindowTitle("学生健康管理系统 by 邓智超 20192121026");
     m.show();
     return app.exec();
 }
```

按照上述的组织构架，就完成了学生健康管理系统的可视化编写。

至此，选做部分结束！

实验思考题解决方案：

1、 如果学生人数超过千万的时候，会出现什么现象，你认为该如何

解决？

答：

当数据超过千万条的时候，若是组织在同一文件，则文件的读取和写入速度都会大幅下降。对于链表的遍历，也会变得困难起来，大大增加了执行操做的时间成本。

解决方案：

① 可以考虑分文件组织存储，如千万条的数据，拆成 100 个文件，每个文件存储 10 万条数据。对于函数的调用，多一个 flag 标签，不同的标签对应对不同的文件进行操作。

② 考虑将存储方式由文件读写改成数据库，它的存储空间很大，可以存放百万条、千万条、上亿条数据。它能更合适的组织数据、更方便的维护数据、更严密的控制数据和更有效的利用数据。

2、 如果要实现一个商品进销存管理系统，你觉得该如何解决？

解决方案：

类似于本次实验的学生健康管理系统，创建一个 fruit 类，其属性有名称，数量，价格。再创建一个 fruitData 类，以链表的形式存储各类水果的数据，并为其添加销售水果，进货，调整价格，退货等操作。此外，还可以增加顾客的信息，新建 customer 类和 customerData 类，customer 类存储会员信息，customerData 类以链表形式存储用户信息，可执行会员增加、删除、会员购买水果，查询消费记录，退货等操作。在 main 函数中，显示菜单并根据用户输入执行相应的操作。具体的实现细节与学生健康管理系统都是一样的。

3、 重新检视一下自己所编制的源程序，分析是否可以进一步让程序变得精简。

重新阅读自己写的代码后，我发现添加，删除，插入，查找这几个函数都需要判断数据是否存在已经获取用户输入的数据。我写的代码每次都是重新码代码的，这就造成了篇幅过长。为此我认为可以写两个函数来使得代码精简化。isRepeat()函数用于判断是否存在数据重复，若重复则输出提示信息。getAndSetInput()函数

用于获得输入并新建和赋值节点，返回节点地址。具体代码如下:

```
int studentData::isRepeat(string name, string number)
{
    student* current=first;
    if (first == 0)return 0;//空链表不可能重复
    if (searchName(name) != 0)//searchName()函数已定义过
    {
        cout << "姓名重复" << endl;
        return 1;//1 表示重复
    }
    if (searchNum(number, 0) != 0)//searchNum()已定义过
    {
        cout << "学号" << endl;
        return 1;//1 表示重复
    }
    return 0;
}

student* p studentData::getAndSetInput()
{
    string name;
    string number;
    string gender;
    string dateOfBirth;
    string healthCondition;
    student* p = new student;
    p -> addStudent(name, number, gender, dateOfBirth, healthCondition);//设置属性值，该函数之前已定义
    return p;
```

}

审视 Qt 可视化界面的代码，发现其组织结构比较乱，很多代码也发生了重复。由于刚刚接触 Qt 没多久，我对于其使用方法还不是很清楚，为了实现系统功能，初步版本的代码逻辑较乱，后续将持续改进代码。

四、实验总结（心得体会）

通过此次实验，我对链表形式的数据组织方式有了更深的理解。进一步熟悉和掌握了 VC 环境下的编译、调试和执行的方法及步骤。也熟悉线性表链式存储的实现方式及其应用。上学期有一个小小图书馆的项目，其实现和本次实验差不多，当时我还没有太弄懂为什么是这样做的，这次实验，让我懂得了基本的思路原理，相信下一次遇到此类题目，我定不会无从下手了！

不足之处：

① 审阅代码后发现，重复的部分较多，之后可以考虑写成函数的形式。对于代码的简洁性，还有待改进。（2020.10.25）

② 对于 Qt 平台的使用不够熟练，学生健康管理系统可视化的实现代码写的比较乱，代码复用率较低，重复部分较多。之后将逐步改进。（2020.11.08）

③ CUI 界面和 GUI 界面的实现都是使用 asc11 文件来存储数据的，这可能会导致信息安全问题，后续的版本将采用二进制文件存储的方法以改进这一缺陷。（2020.11.08）

五、参考文献

版权声明怎么写：

https://blog.csdn.net/wang5969359/article/details/11825389

数据库简介：

https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93/103728

进销存管理系统：

https://www.cnblogs.com/wzjhoutai/p/7128118.html

《C++ Coding Standard》
《高质量 C++/C 编程指南》
《数据结构》教材