

学生学号	0121910870616	实验课成绩	
------	---------------	-------	--

武汉理工大学

学生实验报告书

实验课程名称	数据结构
开 课 学 院	计算机科学与技术学院
指导教师姓名	李晓红
学 生 姓 名	邓晟淼
学生专业班级	软件工程 1902

2020 -- 2021 学年 第 1 学期

课程名称：数据结构

实验项目名称	基于单链表实现学生信息管理系统			实验成绩	
实验者	邓晟淼	专业班级	软件 1902	实验日期	2020 年 10 月 28 日

第一部分 实验目的和内容

一、实验目的

掌握线性表的链式存储结构 单链表的 定义及其基本运算 的 C 语言实现。

二、实验内容

项目名称：学生信息管理系统

项目内容：设计一个学生信息管理系统，实现对学生基本信息的添加、删除、修改和查询等

操作， 其中每个学生信息包含学号，姓名和绩点。要求系统完成以下主要功能：

- (1) 显示：显示当前所有学生信息记录；
- (2) 录入：从键盘输入一条学生信息记录，插入到表中指定的位置；
- (3) 查找：根据学号或者记录的位置查找学生的各项信息；
- (4) 删除：删除指定位置的学生信息记录；
- (5) 更新：更新指定位置的学生信息记录；
- (6) 统计：统计表中学生人数。
- (7) 排序：按照学号或者绩点进行排序
- (8) 清空：清空表中所有记录

三、主要仪器设备及耗材

PC；

Microsoft Visual Studio；

第二部分 实验过程

一、实验设计

1. 描述该学生管理系统的结构框架和设计，该系统由 4 部分组成。

2. Node.h 头文件描述了链表结点的结构体组成，在其中使用到了 c++ 的模板类去描述 Node 结点的数据域，通过模板类使得该链表具有可拓展性，能够运用到各种数据结构中，具有通用性，可重复性的特点。

3. LinkList.h 头文件描述了链表对象的功能实现，实现了实验要求中提到的所有功能。

LinkList 对象的数据成员由头指针 *head 和 int 型链表长度变量 length 构成。成员函数包括：① LinkList 的无参数构造和有参数构造，无参数构造将构造一个空结点使头指针指向该节点，而有参数构造需要提供 ElemType 型的数组以及数组长度，使用类中的 Insert() 函数进行插入构造；② LinkList 的析构函数。使用该析构函数，可以使链表构造时 new 的结点空间被释放掉，从而将该 LinkList 对象析构；③ GetLength, SetLength, GetHead 函数，前两个函数可以返回和设置 length 的值。最后一个函数可以返回链表的长度和链表的头指针，方便在之后的 Student.h 中进行调用；④ IsEmpty 函数，这个函数可以判断该单链表是否为空（虽然我还是经常直接用 head->next == nullptr 进行判断.....）；⑤ Locate 函数可以进行元素的定位，在给定了一个元素之后可以直接将该元素的索引位置返回到主调函数。⑥ Get 和 Set 函数可以获得指定位置的数据域元素，或者设置索引处的数据域的值。⑦ Delete 函数可以将索引处结点的删除，再将链表链接上。⑧ Insert 函数可以在指定位置插入元素，允许输入 0-n 的值进行插入，其会自动判断插入位置是否合法，是否溢出，若溢出则会自动插入到链表的表尾。

4. Student.h 头文件则是定义了 student 结构体，将这个结构体作为模板类中的 ElemType 模板以实现可重复性的功能，student 结构体中，包含了 int 型的学号 num, string 类型的姓名 name, float 类型的绩点 grade，用以实现学生信息管理系统。在该头文件中，我使用了 LinkList.h 中的链表类，以实现链表的功能，我定义了如下的函数接口以实现增删改查排等功能：① ShowData 函数用以实现学生信息的输出，在调用该函数后，将获取链表的头节点，并遍历该链表，输入相关的学生信息，以及全部记录的学生人数。② InsertDate 函数用以实现录入学生信息的功能，通过调用该函数，输入学生的信息，以及插入链表的位置，将调用链表的成员函数 Insert 来进行插入结点。③ FindData 函数允许通过输入学生的学号，或者是学生在链表中的位置，以遍历链表，在遍历完成后将输出学生的各项信息。④ DeleteData 函数可以删除指定链表位置的结点。通过调用链表的成员函数 Delete 以达到删除结点的目的。⑤ RefreshData 函数将更新指定位置的学生信息，在输入需要更新的位置以及节点信息，将调用成员函数的 Set 以进行更新⑥ StatisticData 函数将统计学生人数，调用 GetLength 以进行输出⑦ SortData 函数将进行链表的排序，该排序算法通过冒泡排序算法对链表进行排序，以得到排序后的链表。该算法允许对学号进行排序或者对绩点进行排序⑧ ClearData 函数将 LinkList 对象进行清除，将头节点指向空指针，将长度置零。

5. demo.cpp 将调用 Student.h 中的各接口，以实现功能。以上为该实验的实验设计。

二、实验代码

```
#pragma once
//Node.h
#include<iostream>
#include<iomanip>
#include<Windows.h>
#include<string>
using namespace std;

template<class Elemtype> struct Node {
    Elemtype data;          //数据域
    Node<Elemtype>* next;    //指针域
};

#pragma once
#include "Node.h"
//LinkList.h
template <class ElemType>
class LinkList {
protected:
    //单链表的数据成员
    Node<ElemType>* head; // 头结点指针
    int length; // 单链表长度
public:
    //单链表的函数成员
    LinkList();          //无参数的构造函数
    LinkList(ElemType v[], int n);    //有参数的构造函数
    virtual ~LinkList(); //析构函数
    int GetLength() const;    //求单链表长度
    void SetLength(int length);    //设置链表长度
    Node<ElemType>* GetHead();    //返回头节点指针变量
    bool IsEmpty() const;    //判断单链表是否为空
    int Locate(const ElemType& e) const;    //元素定位
    int Get(int index, ElemType& e);    //求指定位置的元素
    int Set(int index, const ElemType& e);    //设置指定位置的元素值
    int Delete(int index, ElemType& e);    //删除元素
    int Insert(int index, const ElemType& e);    //在制定位置插入元素
};

//[函数]    LinkList::LinkList
//[功能]    默认构造函数
```

```

//[[参数]    void
//[[返回]    void
template<class ElemType> LinkList<ElemType>::LinkList() :length(0), head(new Node<ElemType>)
{
    head->next = nullptr;
}

//[[函数]    LinkList::LinkList
//[[功能]    默认构造函数
//[[参数]    ElemType v[], int n(包含了数据个数以及插入的数据)
//[[返回]    void
template<class ElemType> LinkList<ElemType>::LinkList(ElemType v[], int n) :length(0),
head(new Node<ElemType>) {
    head->next = nullptr;
    for (int i = 0; i < n; i++) {
        if ((this->Insert(0, v[i])) == 0) exit(0);    //插入失败，报错0;
    }
}

//[[函数]    LinkList::~LinkList
//[[功能]    析构函数
//[[参数]    void
//[[返回]    void
template<class ElemType> LinkList<ElemType>::~LinkList() {
    if (head->next == nullptr) {
        delete head;
        return;
    }
    Node<ElemType>* temp1 = head, * temp2 = head->next;
    while (temp2 != nullptr) {
        delete temp1;    //删除前结点
        temp1 = temp2;
        temp2 = temp2->next;    //后结点向后移动到为空
    }
    delete temp1;    //将temp1删除
}

//[[函数]    LinkList::SetLength
//[[功能]    设置长度
//[[参数]    int Length
//[[返回]    void
template<class ElemType> void LinkList<ElemType>::SetLength(int length) {
    this->length = length;
}

```

```

//[函数]    LinkList::GetHead
//[功能]    //返回头节点指针变量
//[参数]    void
//[返回]    Node<ElemType>
template<class ElemType> Node<ElemType>* LinkList<ElemType>::GetHead() {
    return head;
}

//[函数]    LinkList::Insert
//[功能]    //在制定位置插入元素
//[参数]    int index:插入位置    const ElemType &e:插入数据
//[返回]    int TRUE表示成功 FALSE表示失败
template<class ElemType> int LinkList<ElemType>::Insert(int index, const ElemType& e) {
    Node<ElemType>* temp, * temp1, * tpr;    //temp指针用于移动位置, tpr用于申请空间
    tpr = nullptr;
    tpr = new Node<ElemType>;    //申请指向Node结点的指针
    if (tpr == nullptr) return 0;    //分配空间失败
    tpr->data = e;
    if (index >= length) {
        if (index > length) {
            cout << "欲插入位置索引数大于链表长度, 将插入到表尾!" << endl;
            system("pause");
            system("cls");
        }
        temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = tpr;
        tpr->next = nullptr;
    }
    else if (index == 0 || length == 0) {    //插入表头
        temp = head->next;
        tpr->next = temp;
        head->next = tpr;
    }
    else if (index < length && index > 0) {
        temp = head;
        for (int i = 0; i < index; i++) {
            temp = temp->next;    //将指向index的前一个结点
        }
        temp1 = temp->next;    //指向index的下一个节点
        temp->next = tpr;
    }
}

```

```

        tpr->next = templ;
    }
    length++;          //链表长度加1
    return 1;
}

//[函数]      LinkList::GetLength
//[功能]      求链表的长度
//[参数]      void
//[返回]      int 返回length长度
template<class ElemType> int LinkList<ElemType>::GetLength() const {
    return this->length;
}

//[函数]      LinkList::IsEmpty
//[功能]      判断单链表是否为空
//[参数]      void
//[返回]      bool TRUE空 FALSE非空
template<class ElemType> bool LinkList<ElemType>::IsEmpty() const {
    if (head->next == nullptr) return true;
    else return false;
}

//[函数]      LinkList::Locate
//[功能]      根据元素定位
//[参数]      const ElemType &e    元素值
//[返回]      int 返回定位位置，若为-1表示未找到数据
template<class ElemType> int LinkList<ElemType>::Locate(const ElemType& e) const {
    Node<ElemType>* temp = head;
    int num = 0;        //计数用于返回位置
    while (num < this->length) {
        temp = temp->next;
        num++;
        if (temp->data == e) return num;
    }
    return -1;        //未找到数据
}

//[函数]      LinkList::Get
//[功能]      求指定位置的元素
//[参数]      int index: 指定位置, ElemType& e: 返回元素
//[返回]      int 1表示找到, 0表示未找到
template<class ElemType> int LinkList<ElemType>::Get(int index, ElemType& e) {
    if (index > length) return 0;        //未找到数据

```

```

Node<ElemType>* temp = head;
for (int i = 0; i < index; i++) {
    temp = temp->next;
}
e = temp->data;
return 1;
}

//[函数]    LinkList::Set
//[功能]    设置指定位置的元素值
//[参数]    int index: 指定位置, ElemType& e: 设置元素
//[返回]    int 1表示成功设置, 0表示未设置
template<class ElemType> int LinkList<ElemType>::Set(int index, const ElemType& e) {
    if (index > length || index <= 0) {
        cout << "查找位置失败! ";
        system("pause");
        system("cls");
        return 0;        //未找到数据
    }
    Node<ElemType>* temp = head;
    for (int i = 0; i < index; i++) {
        temp = temp->next;
    }
    temp->data = e;
    return 1;
}

//[函数]    LinkList::Delete
//[功能]    删除指定元素
//[参数]    int index: 指定位置, ElemType& e: 返回元素
//[返回]    int 1表示成功, 0表示失败
template<class ElemType> int LinkList<ElemType>::Delete(int index, ElemType& e) {
    if (length == 0 || index > length) {
        cout << "删除失败!链表为空或值溢出!";
        system("pause");
        system("cls");
        return 0;        //链表为空或溢出
    }
    Node<ElemType>* temp;
    temp = head;        //指向头节点
    for (int i = 0; i < index - 1; i++) {        //指向欲删除的元素的前继
        temp = temp->next;
    }
    temp->next = temp->next->next;

```



```

        this->length--;
        return 1;          //删除成功
    }

//Student.h
#pragma once
#include "LinkList.h"
typedef struct student {
    int num;          //学号
    string name;      //姓名
    float grade;      //绩点
} student;

bool ShowData(LinkList<student>& stu);          //显示学生信息
bool InsertData(LinkList<student>& stu);          //录入学生信息
bool FindData(LinkList<student>& stu);          //根据位置或学号查找学生的各项信息
bool DeleteData(LinkList<student>& stu);          //删除指定位置的学生信息记录
bool RefreshData(LinkList<student>& stu);          //更新指定位置学生信息
bool StatisticData(LinkList<student>& stu);          //统计表中学生人数
bool SortData(LinkList<student>& stu);          //按照学号或者绩点进行排序
bool ClearData(LinkList<student>& stu);          //清空所有信息

bool ShowData(LinkList<student>& stu) {          //显示学生信息
    Node<student>* temp = stu.GetHead();
    // if (true == stu.IsEmpty()) return false;          //若链表为空返回false
    cout << "学生信息表格如下所示:" << endl;
    cout << "-----" << endl;
    cout << std::left << setw(15) << "学号" << setw(15) << "姓名" << setw(15) << "绩点" << endl;
    for (int i = 0; i < stu.GetLength(); i++) {
        temp = temp->next;
        cout << std::left << setw(15) << temp->data.num << setw(15) << temp->data.name <<
        setw(15) << fixed << setprecision(3) << temp->data.grade << endl;
    }
    cout << "-----" << endl;
    cout << "学生信息共有" << stu.GetLength() << "人数据" << endl;
    system("pause");
    system("cls");
    return true;
}

bool InsertData(LinkList<student>& stu) {          //录入学生信息
    student temp_stu;          //临时存放数据;

```

```

    int num;
    cout << "请输入欲插入数据的位置(0-n):";
    cin >> num;
    cin.clear();
    cin.sync();
    cout << "请输入学生学号:";
    cin >> temp_stu.num;
    cin.clear();
    cin.sync();
    cout << "请输入学生姓名:";
    cin >> temp_stu.name;
    cin.clear();
    cin.sync();
    cout << "请输入学生绩点:";
    cin >> temp_stu.grade;
    cout << endl;
    cin.clear();
    cin.sync();
    system("cls");
    stu.Insert(num, temp_stu);
    return true;
}

bool FindData(LinkList<student>& stu) {    //根据位置或学号查找学生的各项信息
    int nSelection, nNumber, nIndex;
    Node<student>* temp = stu.GetHead();
step1:
    cout << "请选择使用学号(0)或位置(1)搜索学生全部信息:";
    cin >> nSelection;
    cin.clear();
    cin.sync();
    system("cls");
    switch (nSelection) {
    case 0: {
        cout << "请输入该学生学号:";
        cin >> nNumber;
        cin.clear();
        cin.sync();
        system("cls");
        for (int i = 0; i < stu.GetLength(); i++) {
            temp = temp->next;
            if (nNumber == temp->data.num) {
                cout << "该学生学号为:" << temp->data.num << "姓名为:" << temp->data.name
                << "绩点为:" << temp->data.grade << endl;
            }
        }
    }
    }
}

```

```

        system("pause");
        system("cls");
        return true;
    }
}
break;
}
case 1: {
    cout << "请输入该学生在链表中的位置:";
    cin >> nIndex;
    cin.clear();
    cin.sync();
    system("cls");
    if (nIndex > stu.GetLength())break;
    for (int i = 0; i < nIndex; i++) {
        temp = temp->next;
    }
    cout << "该学生学号为:" << temp->data.num << "姓名为:" << temp->data.name << "绩点
为:" << temp->data.grade << endl;
    system("pause");
    system("cls");
    return true;
}
default: {
    cout << "无该选项，请重新输入！" << endl;
    system("pause");
    system("cls");
    goto step1;
}
}
cout << "未找到该学生信息" << endl;
system("pause");
system("cls");
return false;
}

bool DeleteData(LinkList<student>& stu) { //删除指定位置的学生信息记录
    int num;
    student temp;
    cout << "请输入删除的指定位置(1-n):";
    cin >> num;
    cin.clear();
    cin.sync();
    system("cls");

```

```

        if (stu.Delete(num, temp)) return true;
        else return false;
    }

bool RefreshData(LinkList<student>& stu) {           //更新指定位置学生信息
    student temp_stu;           //临时存放数据;
    int num;
    cout << "请输入欲更新数据的位置(1-n):";
    cin >> num;
    cin.clear();
    cin.sync();
    cout << "请输入学生学号:";
    cin >> temp_stu.num;
    cin.clear();
    cin.sync();
    cout << "请输入学生姓名:";
    cin >> temp_stu.name;
    cin.clear();
    cin.sync();
    cout << "请输入学生绩点:";
    cin >> temp_stu.grade;
    cin.clear();
    cin.sync();
    system("cls");
    if (stu.Set(num, temp_stu)) return true;
    else return false;
}

bool StatisticData(LinkList<student>& stu) {         //统计表中学生人数
    cout << "学生人数总共有:" << stu.GetLength() << "人" << endl;
    system("pause");
    system("cls");
    return true;
}

bool SortData(LinkList<student>& stu) {              //按照学号或者绩点进行排序
    Node<student>* temp1, * temp2, * head;          //用于遍历stu,存取头节点
    head = stu.GetHead();
    int nSelection = 0;
    cout << "1. 按成绩从高到低" << endl << "2. 按成绩从低到高" << endl << "3. 按学号从高到低"
    << endl << "4. 按学号从低到高" << endl << "请输入排序方式:";
    cin >> nSelection;
    cin.clear();
    cin.sync();

```

```

system("cls");
if (head->next == nullptr || head->next->next == nullptr) return false;
for (int i = 0; i < stu.GetLength() - 1; i++) {
    temp1 = head;
    temp2 = head->next;
    for (int j = 0; j < stu.GetLength() - i - 1; j++) {
        switch (nSelection) {
            case 1: {
                if (temp2->data.grade < temp2->next->data.grade) { //冒泡排序，排
序为成绩从大到小。
                    temp1->next = temp2->next;
                    temp2->next = temp2->next->next;
                    temp1->next->next = temp2;
                    //移动，指针变化temp2向后移动一位
                    temp1 = temp1->next;
                }
                else { //未移动，指针向后位移一位
                    temp1 = temp2;
                    temp2 = temp2->next;
                }
                break;
            }
            case 2: {
                if (temp2->data.grade > temp2->next->data.grade) { //冒泡排序，排
序为成绩从小到大。
                    temp1->next = temp2->next;
                    temp2->next = temp2->next->next;
                    temp1->next->next = temp2;
                    //移动，指针变化temp2向后移动一位
                    temp1 = temp1->next;
                }
                else { //未移动，指针向后位移一位
                    temp1 = temp2;
                    temp2 = temp2->next;
                }
                break;
            }
            case 3: {
                if (temp2->data.num < temp2->next->data.num) { //冒泡排序，排序为序
号从大到小。
                    temp1->next = temp2->next;
                    temp2->next = temp2->next->next;
                    temp1->next->next = temp2;
                    //移动，指针变化temp2向后移动一位

```

```

        temp1 = temp1->next;
    }
    else {          //未移动，指针向后位移一位
        temp1 = temp2;
        temp2 = temp2->next;
    }
    break;
}
case 4: {
    if (temp2->data.num > temp2->next->data.num) {          //冒泡排序，排序为序
号从小到大。
        temp1->next = temp2->next;
        temp2->next = temp2->next->next;
        temp1->next->next = temp2;
        //移动，指针变化temp2向后移动一位
        temp1 = temp1->next;
    }
    else {          //未移动，指针向后位移一位
        temp1 = temp2;
        temp2 = temp2->next;
    }
    break;
}
default: return false;
}
}
return true;
}

bool ClearData(LinkList<student>& stu) {          //清空所有信息
    stu.GetHead()->next = nullptr;
    stu.SetLength(0);
    return true;
}

//Demo.h
#include "Student.h"
#include "LinkList.h"
int main() {
    LinkList<student> stu;
    int nSelection;

```

```
do {  
    cout << "欢迎使用学生信息管理系统!" << endl;  
    cout << "1. 显示学生信息" << endl;  
    cout << "2. 录入学生信息" << endl;  
    cout << "3. 查找学生信息" << endl;  
    cout << "4. 删除学生信息" << endl;  
    cout << "5. 更新学生信息" << endl;  
    cout << "6. 统计学生信息" << endl;  
    cout << "7. 排序学生信息" << endl;  
    cout << "8. 清空学生信息" << endl;  
    cout << "0. 按任意键退出" << endl;  
    cout << "请输入执行的操作:";  
    cin >> nSelection;  
    cin.clear();  
    cin.sync();  
    system("cls");  
    switch (nSelection) {  
        case 1: {  
            ShowData(stu);  
            break;  
        }  
        case 2: {  
            InsertData(stu);  
            break;  
        }  
        case 3: {  
            FindData(stu);  
            break;  
        }  
        case 4: {  
            DeleteData(stu);  
            break;  
        }  
        case 5: {  
            RefreshData(stu);  
            break;  
        }  
        case 6: {  
            StatisticData(stu);  
            break;  
        }  
        case 7: {  
            SortData(stu);  
            break;  
        }  
    }  
}
```

```

    }
    case 8: {
        ClearData(stu);
        break;
    }
    default: {
        nSelection = 0;
        break;
    }
}
} while (nSelection != 0);

return 0;
}

```

第三部分 实验结果及分析

对实验结果进行分析。

下面将进行截图演示各种功能的运行结果：

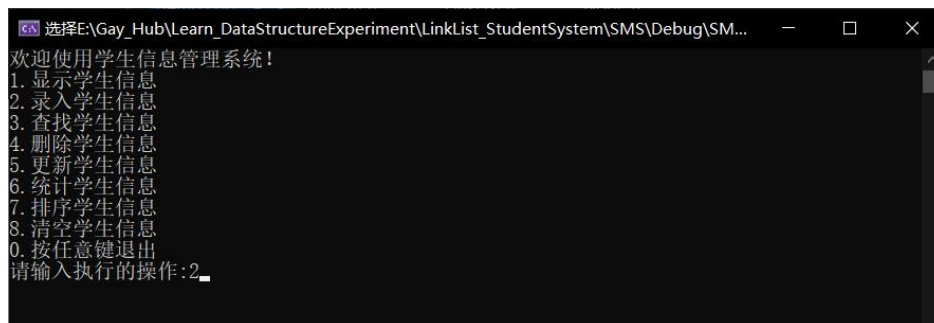


图 1.执行录入学生信息功能

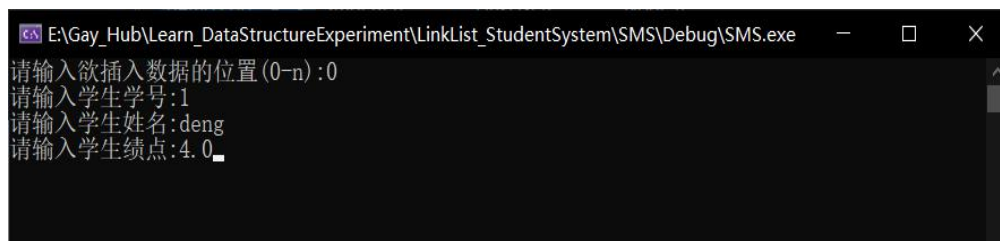


图 2.录入的学生信息

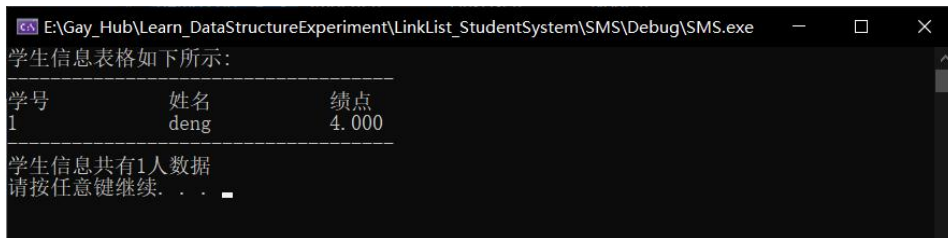


图 3.显示学生信息

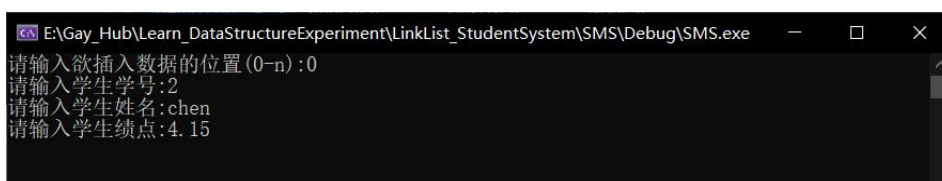


图 4.在 0 号位置再插入一个数据，该数据应在学号 1 之前

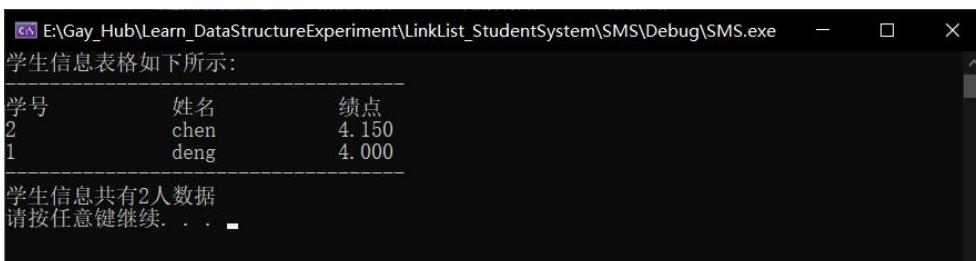


图 5.如图所示，图 4 的录入结果及位置正确

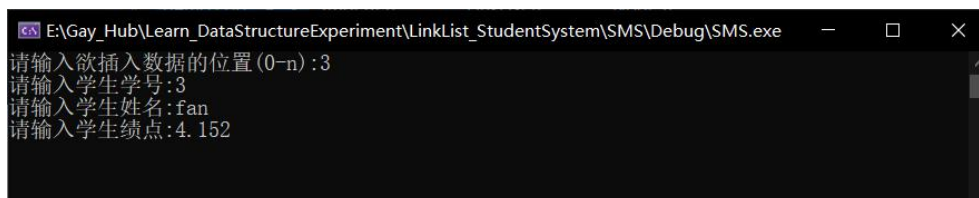


图 6.插入位置 3，该位置溢出了链表的最大值，应该报错并且插入到表尾

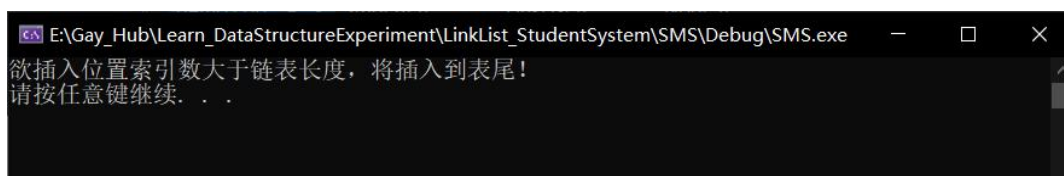


图 7.显示正确，报错

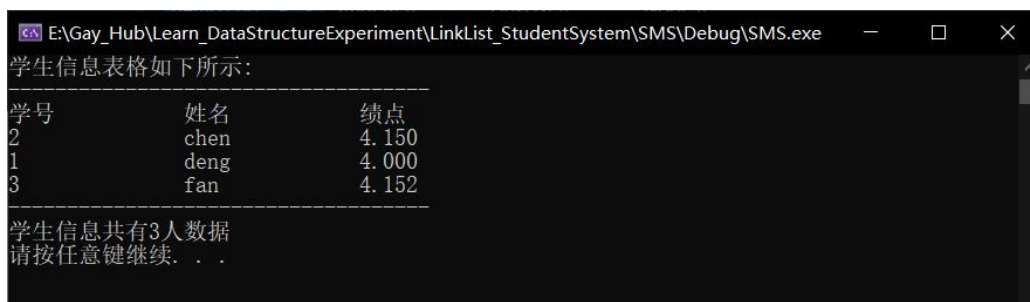
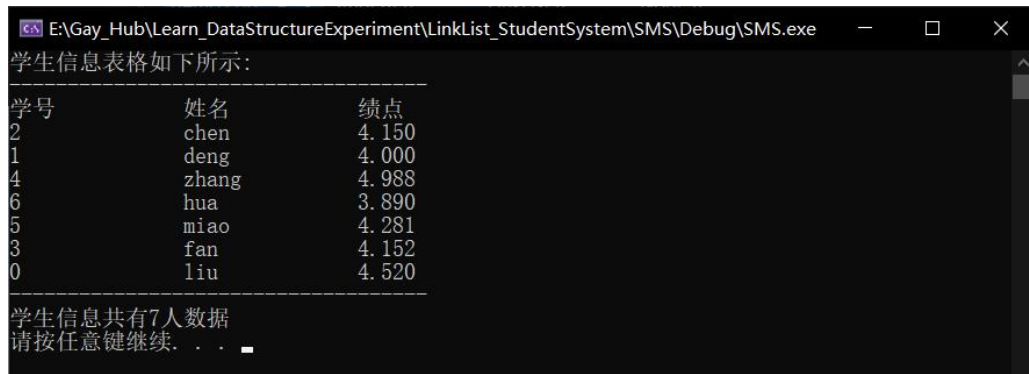
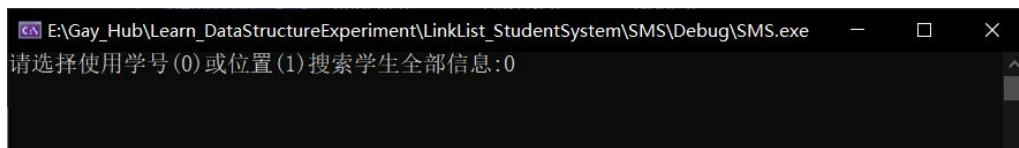


图 8.显示结果，插入到了表尾



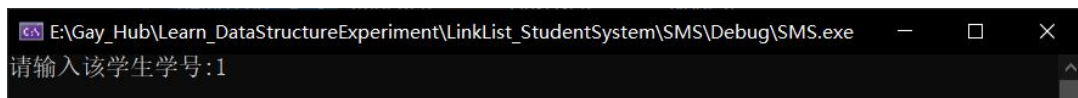
```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SMS.exe
学生信息表格如下所示:
-----
学号      姓名      绩点
2         chen     4.150
1         deng     4.000
4         zhang    4.988
6         hua      3.890
5         miao     4.281
3         fan      4.152
0         liu      4.520
-----
学生信息共有7人数据
请按任意键继续. . .
```

图 9.再多插几个数据，便于之后的排序更美观



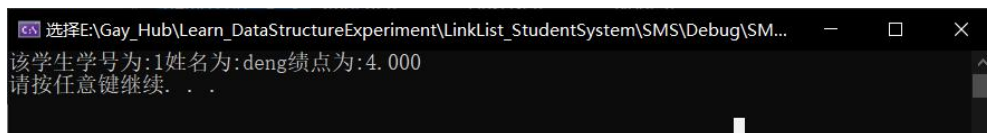
```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SMS.exe
请选择使用学号(0)或位置(1)搜索学生全部信息:0
```

图 10.使用位置查找学生信息



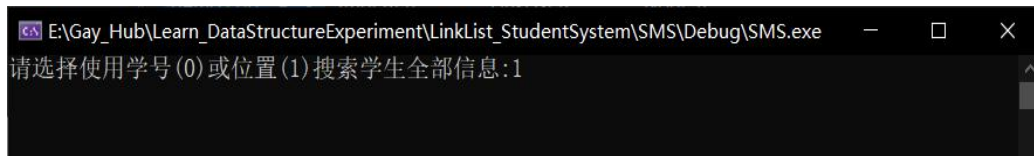
```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SMS.exe
请输入该学生学号:1
```

图 11.查找学号为 1 的学生信息



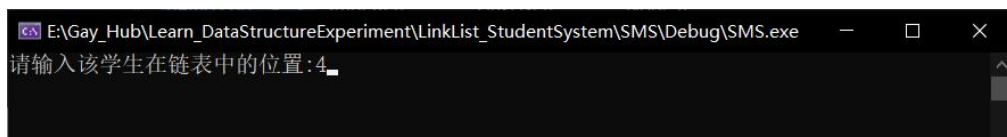
```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SM...
该学生学号为:1姓名为:deng绩点为:4.000
请按任意键继续. . .
```

图 12.查询到的信息



```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SMS.exe
请选择使用学号(0)或位置(1)搜索学生全部信息:1
```

图 13.使用位置搜索



```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SMS.exe
请输入该学生在链表中的位置:4
```

图 14.查询位置 4 的学生信息

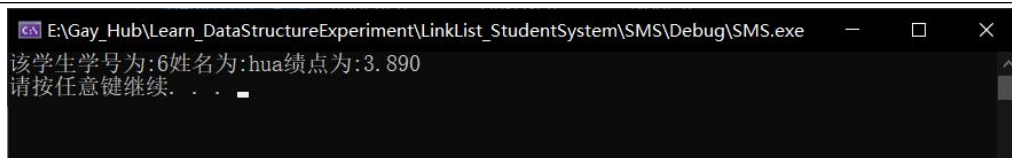


图 15.查询结果正确

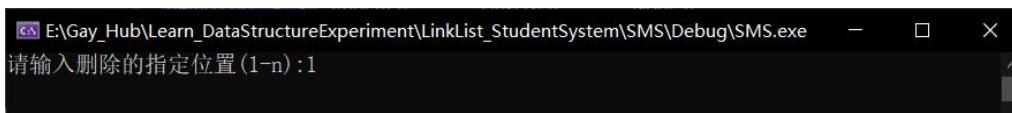


图 16.删除位置 1 的结点

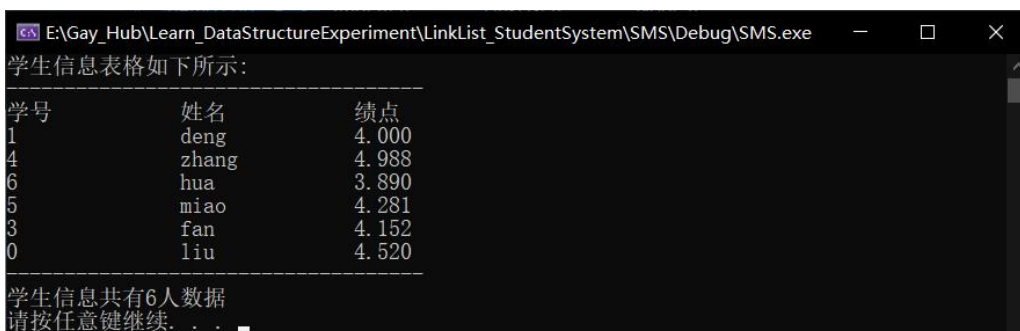


图 17.位置 1 的结点信息已经被删除

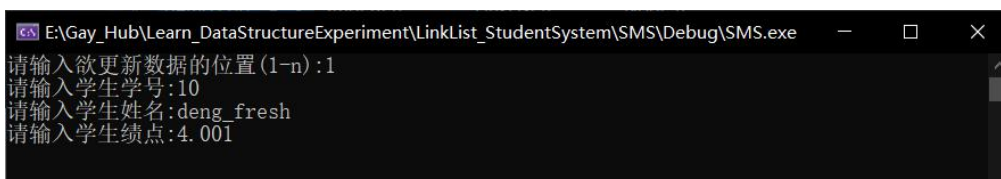


图 18.更新位置 1 的结点信息

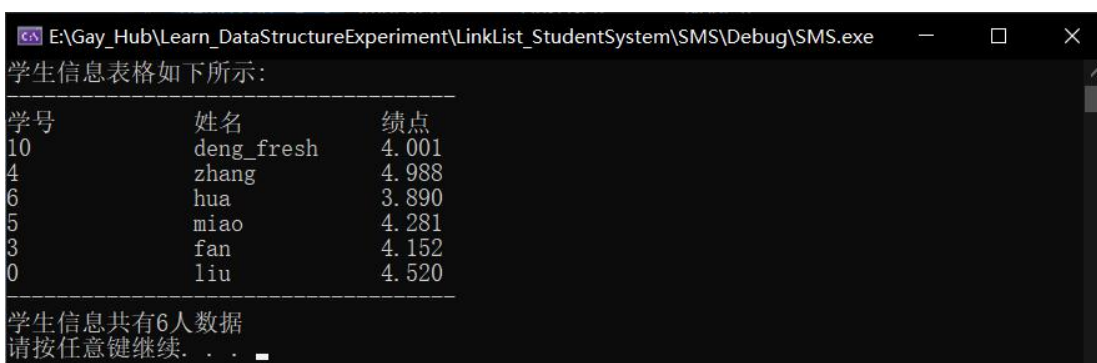


图 19.更新结果

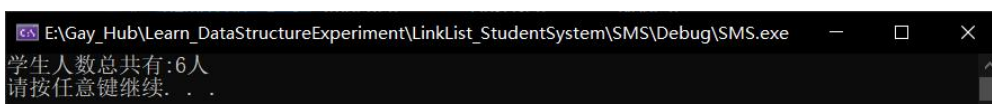


图 20.统计信息，输出总人数

```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SMS.exe
学生信息表格如下所示:
-----
学号      姓名      绩点
4         zhang     4.988
0         liu       4.520
5         miao      4.281
3         fan       4.152
10        deng_fresh 4.001
6         hua       3.890
-----
学生信息共有6人数据
请按任意键继续. . .
```

图 21.将学生的绩点按从高到低排序

```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SMS.exe
学生信息表格如下所示:
-----
学号      姓名      绩点
6         hua       3.890
10        deng_fresh 4.001
3         fan       4.152
5         miao      4.281
0         liu       4.520
4         zhang     4.988
-----
学生信息共有6人数据
请按任意键继续. . .
```

图 22.将学生的绩点按从低到高排序

```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SMS.exe
学生信息表格如下所示:
-----
学号      姓名      绩点
10        deng_fresh 4.001
6         hua       3.890
5         miao      4.281
4         zhang     4.988
3         fan       4.152
0         liu       4.520
-----
学生信息共有6人数据
请按任意键继续. . .
```

图 23.将学生的学号按从高到低排序

```
E:\Gay_Hub\Learn_DataStructureExperiment\LinkList_StudentSystem\SMS\Debug\SMS.exe
学生信息表格如下所示:
-----
学号      姓名      绩点
0         liu       4.520
3         fan       4.152
4         zhang     4.988
5         miao      4.281
6         hua       3.890
10        deng_fresh 4.001
-----
学生信息共有6人数据
请按任意键继续. . .
```

图 24.将学生的学号按从低到高排序

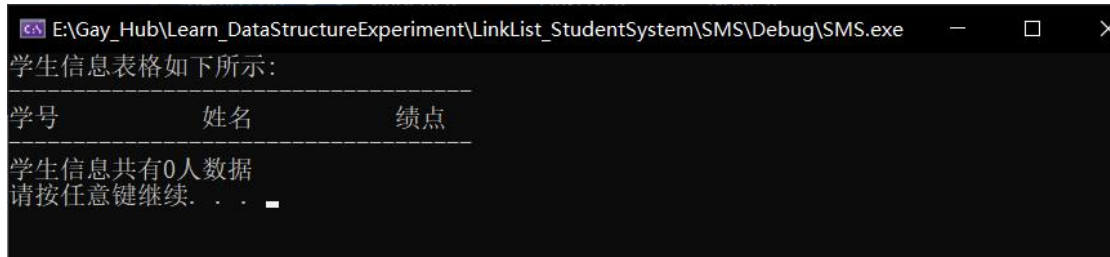


图 25.清空链表



图 26.运行结束

第四部分 实验小结

对此次实验进行总结。

这次实验使用到了单链表以及模板类，通过构造了链表以实现学生信息管理系统的实现，此结构可以允许储存的学生信息随插入的信息增加而增加，除此之外，运用到了模板类的功能使得该链表类能够更加有可移植性，通用性。通过此次实验，我学到了很多新的知识与方法，了解到了各种新方法，在之后的数据结构课程学习中，我一定会继续钻研，认真完成各种数据结构的实现。