

重 庆 交 通 大 学

《 算 法 与 数 据 结 构 》 课 程

实 验 报 告

班 级： 计算机 专业 19 级 1 班

实验项目名称： 链表实验

实验项目性质： 验证性实验

实验所属课程： 数据结构 A

实验室(中心)： B01 409

指 导 教 师： 鲁云平

实验完成时间： 2020 年 10 月 15 日

教师评阅意见：

签名： 年 月 日

实验成绩：

一、实验目的

- 1、实现线性表的链式存储结构（链表）。
- 2、熟悉 C++ 程序的基本结构，掌握程序中的头文件、实现文件和主文件之间的相互关系及各自的作用。
- 3、熟悉链表的基本操作方式, 掌握链表相关操作的具体实现。

二、实验内容及要求

实验内容

对链式存储结构的线性表进行一些基本操作。主要包括：

（1）插入：操作方式为在指定元素前插入、在指定元素之后插入、在指定位置完成插入

（2）删除：操作方式可分为删除指定元素、删除指定位置的元素等，尝试实现逻辑删除操作。

（3）显示数据

（4）查找：查询指定的元素（可根据某个数据成员完成查询操作）

（5）定位操作：定位指定元素的序号

（6）更新：修改指定元素的数据

（7）数据文件的读写操作等。

三、系统分析

(1) 数据方面:

链表类 (LinkedList) 中包含链节类 (LinkNode) 的指针, 包括头节点 (first) 和尾节点 (last), 链节中生成链节指针 (即 `LinkNode<T>*`), 用于将链节连接起来, 构成链表。

使用模板 (template) 构建链表, 可以使用任意数据类型。

(2) 功能方面:

链表 (LinkedList) 大致包含以下几种功能 (对外接口):

1. 插入数据 (在指定位置插入; 在指定元素前插入; 在指定元素后插入; 在最后插入)
2. 查询元素 (查找元素下标; 定位元素)
3. 打印所有元素
4. 删除元素 (删除所有指定元素; 删除指定位置元素)
5. 修改元素 (修改指定位置元素; 将所有元素 A 修改为元素 B)
6. 返回指定位置的元素
7. 返回目前元素个数
8. 返回目前所能容纳的元素个数
9. 判断顺序表是否为空
10. 清空顺序表
11. 查找某个元素的个数
12. 从文件中读取相同类型数据覆盖顺序表 (读文件)
13. 将元素数据写入文件 (写文件)

四、系统设计

(1) 设计的主要思路

- ① 定义 LinkedList 模板
- ② 实现 LinkedList 模板功能
- ③ 设计 MeneLinkedList 类, 即链表菜单类, 在该类中调用 LinkedList 对象, 包括 `LinkedList<int>` 对象和 `LinkedList<Student>` 对象 (Student 为自定义类型)。
- ④ 在 main 函数中使用 MeneLinkedList 类生成对象, 调出菜单。

(3) 数据结构的设计

在 LinkNode（链节类）中，包含以下成员

```
T data;    //储存的单个数据
```

```
LinkNode<T>* link; //下一节
```

其中 data 储存数据；link 储存下一个链节，以做到链表的链节。

在 LinkedList（链表类中），包含以下成员

```
int length;    //目前容量
```

```
LinkNode<T>* first; //头节点(指向第一个节点)
```

```
LinkNode<T>* last;  //尾结点(指向最后一个节点)
```

其中 first 和 last 分别为头结点和尾结点，分别指向第一个和最后一个链节，不储存数据，用于链表的获取。其中 last 也主要用于尾部新链节的插入。

Length 用于记录元素个数，即除头尾结点的链节个数。

(4) 基本操作的设计

链表的基本操作（方法）：

private:

```
LinkNode<T>* FindByPos(int pos);    //返回 pos 下标的节点
```

```
LinkNode<T>* FindBeforePos(int pos); //返回 pos 下标前的节点,用于插入数据
```

```
LinkNode<T>* FindEleBefore(T ele);  //通过元素找之前的节点
```

```
LinkNode<T>* FindEle(T ele);        //通过元素找该节点
```

```
void MoveLast(); //移动尾结点
```

public:

```
LinkedList();    //构造函数
```

```
~LinkedList();  //析构函数
```

```
bool Insert(int pos, T ele); //指定位置插入数据
```

```
void PushBack(T ele);        //在最后插入////////未完成
```

```
bool InsertBefore(T eleBefore, T ele); //在指定元素前插入元素
```

```
bool InsertAfter(T eleAfter, T ele);   //在指定元素后插入元素
```

```
int Search(T ele); //找元素的下标(不存在为-1)
```

```

int Locate(T ele); //找元素的位置（下标+1，不存在为0）
void Display();    //显示所有数据

void Erase(T ele);      //删除指定元素(所有)
bool EraseByPos(int pos); //删除指定位置元素
void Clear(); //清空元素

int Size(); //元素个数

bool Modify(int pos, T ele); //修改指定位置的元素
void ModifyAToB(T eleA, T eleB); //讲所有元素 A 修改为元素 B

void ReadFile(string fileName); //读文件（相同类型数据）
void WriteFile(string fileName); //写文件（相同类型数据）

```

五、编程环境与实验步骤

（1）编程环境

操作系统：Windows 10

编程工具：Visual Studio 2019

（2）实验步骤

①LinkedList.h 用于定义模板类 LinkedList 和其相关属性和方法，也定义并实现了链节类（LinkNode）。

②LinkedList.cpp 用于实现模板类 LinkedList 的方法。

③Student.h 用于定义类 Student。

④Student.cpp 用于实现类 Student。

⑤MenuLinkedList.h 用于定义 MenuLinkedList 类，其中构建菜单使用模板类 LinkedList。

⑥MenuLinkedList.cpp 用于实现类 MenuLinkedList 的方法

⑦Main.cpp 用于实现 main 函数，在其中生成 MenuLinkedList 对象，调用菜单方法。

（3）编译参数

无。

六、实现代码

在 LinkedList.cpp 文件中实现主要功能

LinkedList.cpp:

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

/// <summary>
/// 查找指定下标的节点
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="pos">下标</param>
/// <returns></returns>
template<class T>
LinkNode<T>* LinkedList<T>::FindByPos(int pos)
{
    if (pos < 0 || pos >= length)
        return NULL;

    LinkNode<T>* n = this->first;
    for (int i = 0; i < pos + 1; i++)
        n = n->link;
    return n;
}

/// <summary>
/// 查找下标前的节点
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="pos">下标</param>
/// <returns></returns>
template<class T>
LinkNode<T>* LinkedList<T>::FindBeforePos(int pos)
{
    if (pos < 0 || pos > length)
        return NULL;

    LinkNode<T>* n = this->first; //查找的下标前的节点
    for (int i = 0; i < pos; i++)
        n = n->link;
```

```

        return n;
    }

    /// <summary>
    /// 查找元素前节点
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="ele">数据</param>
    /// <returns>元素前节点</returns>
    template<class T>
    LinkNode<T>* LinkedList<T>::FindEleBefore(T ele)
    {
        LinkNode<T>* before = this->first;
        while (before->link != NULL) {
            if (before->link->data == ele)
                return before;
            before = before->link;
        }
        return NULL;
    }

```

```

    /// <summary>
    /// 通过元素查找节点
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="ele">查找元素</param>
    /// <returns>指定元素节点</returns>
    template<class T>
    LinkNode<T>* LinkedList<T>::FindEle(T ele)
    {
        LinkNode<T>* current = this->first->link;
        while (current != NULL) {
            if (current->data == ele)
                return current;
            current = current->link;
        }
        return NULL;
    }

```

```

    /// <summary>
    /// 移动尾结点
    /// </summary>
    /// <typeparam name="T"></typeparam>
    template<class T>

```

```

void LinkedList<T>::MoveLast()
{
    if (length == 0) {
        last = first;
        return;
    }

    while (last->link != NULL) {
        last = last->link;
    }
}

template<class T>
LinkedList<T>::LinkedList()
{
    this->first = new LinkNode<T>;
    this->last = first;

    length = 0;
}

template<class T>
LinkedList<T>::~~LinkedList()
{
    this->Clear();
}

/// <summary>
/// 在指定下标插入元素
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="pos">指定下标</param>
/// <param name="ele">插入元素</param>
/// <returns>是否插入成功</returns>
template<class T>
bool LinkedList<T>::Insert(int pos, T ele)
{
    LinkNode<T>* before = FindBeforePos(pos); //插入下标前的节点
    if (before == NULL)
        return false;

    LinkNode<T>* newData = new LinkNode<T>(ele);
    if (newData == NULL) {
        cerr << "内存分配失败" << endl;
    }
}

```



```

        exit(1);
    }

    newData->link = before->link;
    before->link = newData;
    length++;
    MoveLast();
    return true;
}

/// <summary>
/// 在最后添加元素
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="ele">要添加的元素</param>
template<class T>
void LinkedList<T>::PushBack(T ele)
{
    LinkNode<T>* n = new LinkNode<T>(ele);

    last->link = n;
    last = n;
    length++;
}

```

```

/// <summary>
/// 在指定元素前插入元素
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="eleBefore">指定元素</param>
/// <param name="ele">插入元素</param>
/// <returns>是否插入成功</returns>
template<class T>
bool LinkedList<T>::InsertBefore(T eleBefore, T ele)
{
    LinkNode<T>* before = FindEleBefore(eleBefore); //eleBefore 前

```

节点

```

    if (before == NULL)
        return false;
    LinkNode<T>* newData = new LinkNode<T>(ele);
    if (newData == NULL) {
        cerr << "内存分配失败！" << endl;
        exit(1);
    }

```

```

        newData->link = before->link;
        before->link = newData;
        length++;
        return true;
    }

    /// <summary>
    /// 在指定元素后插入元素
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="eleAfter">指定元素</param>
    /// <param name="ele">插入元素</param>
    /// <returns>是否插入成功</returns>
    template<class T>
    bool LinkedList<T>::InsertAfter(T eleAfter, T ele)
    {
        LinkNode<T>* current = FindEle(eleAfter); //eleAfter 节点
        if (current == NULL)
            return false;
        LinkNode<T>* newData = new LinkNode<T>(ele);
        if (newData == NULL) {
            cerr << "内存分配失败! " << endl;
            exit(1);
        }

        newData->link = current->link;
        current->link = newData;
        length++;
        MoveLast();
        return true;
    }

    /// <summary>
    /// 查找元素下标
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="ele">查找的元素</param>
    /// <returns>元素下标(-1 为未找到)</returns>
    template<class T>
    int LinkedList<T>::Search(T ele)
    {
        if (length == 0)
            return -1;

```

```

        ListNode<T>* n = this->first->link;
        int pos = 0;
        while (n != NULL) {
            if (n->data == ele)
                return pos;
            n = n->link;
            pos++;
        }
        return -1;
    }

```

```

    /// <summary>
    /// 定位元素位置（下标+1）
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="ele">定位的元素</param>
    /// <returns>元素位置（下标+1）</returns>
    template<class T>
    int LinkedList<T>::Locate(T ele)
    {
        return Search(ele) + 1;
    }

```

```

    template<class T>
    void LinkedList<T>::Display()
    {
        if (length == 0) {
            cout << "元素为 0" << endl;
            return;
        }
    }

```

```

        ListNode<T>* n = this->first->link;
        while (n != NULL) {
            cout << n->data << " ";
            n = n->link;
        }
        cout << endl;
    }

```

```

    /// <summary>
    /// 删除指定元素（所有）
    /// </summary>
    /// <typeparam name="T"></typeparam>

```

```

/// <param name="ele">要删除的元素</param>
template<class T>
void LinkedList<T>::Erase(T ele)
{
    ListNode<T>* nn = this->first;
    while (nn->link != NULL) {
        if (nn->link->data == ele) {
            ListNode<T>* toDel = nn->link; //要删除的节点
            nn->link = toDel->link;
            delete toDel;
            length--;
            continue;
        }
        nn = nn->link;
    }
    MoveLast();
}

/// <summary>
/// 删除指定位置元素
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="pos">元素位置</param>
/// <returns>是否删除成功</returns>
template<class T>
bool LinkedList<T>::EraseByPos(int pos)
{
    ListNode<T>* before = FindBeforePos(pos);
    if (before == NULL)
        return false;

    ListNode<T>* toDel = before->link; //要删除的元素

    before->link = toDel->link;
    delete toDel;
    length--;
    MoveLast();
    return true;
}

/// <summary>
/// 清空元素
/// </summary>
/// <typeparam name="T"></typeparam>

```

```

template<class T>
void LinkedList<T>::Clear()
{
    this->last = this->first;
    LinkNode<T>* p;
    while (this->first->link != NULL) {
        p = this->first->link;
        this->first->link = p->link;

        delete p;
    }
    length = 0;
}

template<class T>
int LinkedList<T>::Size()
{
    return length;
}

/// <summary>
/// 修改指定位置元素
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="pos">指定位置</param>
/// <param name="ele">修改后元素</param>
/// <returns>是否修改成功</returns>
template<class T>
bool LinkedList<T>::Modify(int pos, T ele)
{
    LinkNode<T>* current = FindByPos(pos); //该节点
    if (current == NULL)
        return false;

    current->data = ele;
    return true;
}

/// <summary>
/// 将所有元素 A 修改为元素 B(没有 A 元素则不修改)
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="eleA">元素 A</param>
/// <param name="eleB">元素 B</param>

```

```

template<class T>
void LinkedList<T>::ModifyAToB(T eleA, T eleB)
{
    LinkNode<T>* current = this->first->link;
    while (current != NULL) {
        if (current->data == eleA)
            current->data = eleB;
        current = current->link;
    }
}

```

```

/// <summary>
/// 读文件，将数据覆盖该链表
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="fileName">文件名</param>
template<class T>
void LinkedList<T>::ReadFile(string fileName)
{
    ifstream f;
    f.open(fileName, ios::in);
    if (!f.is_open()) {
        cerr << "文件打开失败!" << endl;
        exit(1);
    }

    this->Clear();

    int count; //数据个数
    f >> count; //文件第一行为元素个数
    for (int i = 0; i < count; i++) {
        T newData;
        f >> newData;
        this->Insert(i, newData);
    }
    f.close();
}

```

```

template<class T>
void LinkedList<T>::WriteFile(string fileName)
{
    ofstream f;
    f.open(fileName, ios::out);
    if (!f.is_open()) {

```

```

        cerr << "文件打开失败!" << endl;
        exit(1);
    }

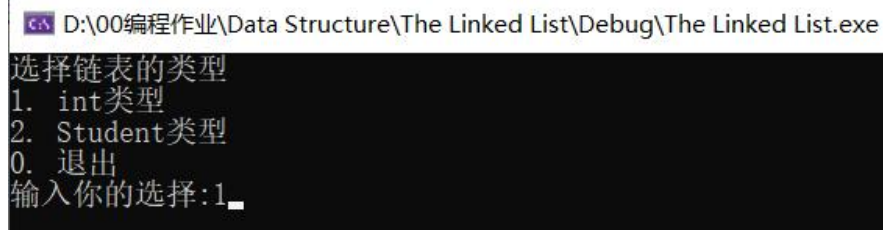
    f << length << endl; //写入元素个数
    LinkNode<T>* current = this->first->link; //用于遍历
    while (current != NULL) {
        f << current->data << endl;
        current = current->link;
    }
    f.close();
}

```

七、测试结果与说明

至少完成功能测试，使用测试数据测试相关功能是否符合设计要求。

1. int 类型测试：

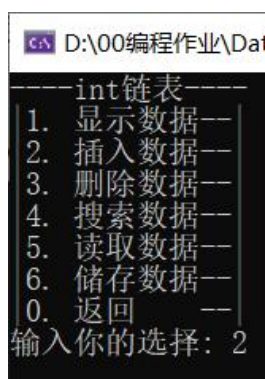


```

C:\> D:\00编程作业\Data Structure\The Linked List\Debug\The Linked List.exe
选择链表的类型
1. int类型
2. Student类型
0. 退出
输入你的选择:1_

```

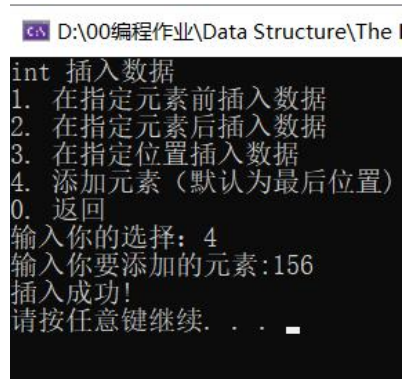
(1) 插入数据



```

D:\00编程作业\Data Structure\The Linked List\Debug\The Linked List.exe
int 链表
1. 显示数据
2. 插入数据
3. 删除数据
4. 搜索数据
5. 读取数据
6. 储存数据
0. 返回
输入你的选择: 2

```



```

D:\00编程作业\Data Structure\The Linked List\Debug\The Linked List.exe
int 插入数据
1. 在指定元素前插入数据
2. 在指定元素后插入数据
3. 在指定位置插入数据
4. 添加元素 (默认为最后位置)
0. 返回
输入你的选择: 4
输入你要添加的元素:156
插入成功!
请按任意键继续. . .

```

```

C:\ D:\00编程作业\Data Structure\The
int 插入数据
1. 在指定元素前插入数据
2. 在指定元素后插入数据
3. 在指定位置插入数据
4. 添加元素（默认为最后位置）
0. 返回
输入你的选择：2
输入你想要插入后元素：156
输入你想要插入元素：20
插入成功！
请按任意键继续. . .

```

```

C:\ D:\00编程作业\Data Structure\The Li
int 插入数据
1. 在指定元素前插入数据
2. 在指定元素后插入数据
3. 在指定位置插入数据
4. 添加元素（默认为最后位置）
0. 返回
输入你的选择：4
输入你要添加的元素：62
插入成功！
请按任意键继续. . .

```

(2) 显示数据

```

C:\ D:\00编程作业\Data S
int链表---
1. 显示数据---
2. 插入数据---
3. 删除数据---
4. 搜索数据---
5. 读取数据---
6. 储存数据---
0. 返回 ---
输入你的选择：1

```

```

C:\ D:\00编程作业\Data Structure\T
156 20 62
请按任意键继续. . .

```

(3) 搜索数据

```

C:\ D:\00编程作业\Data Str
int链表---
1. 显示数据---
2. 插入数据---
3. 删除数据---
4. 搜索数据---
5. 读取数据---
6. 储存数据---
0. 返回 ---
输入你的选择：4

```

```

C:\ D:\00编程作业\Data Structur
int 查询数据
1. 查找某元素位置
2. 修改指定位置元素
3. 将所有元素A改为B
0. 返回
输入你的选择：1
输入你想查询的元素：20
该元素位置为：2
请按任意键继续. . .

```

(4) 删除数据

```

C:\ D:\00编程作业\Data Str
int链表---
1. 显示数据---
2. 插入数据---
3. 删除数据---
4. 搜索数据---
5. 读取数据---
6. 储存数据---
0. 返回 ---
输入你的选择：3

```

```

C:\ D:\00编程作业\Data Structure\The Link
int 删除数据
1. 删除指定元素(所有)
2. 删除指定位置元素
0. 返回
输入你的选择：1
输入你想要删除的元素：156
删除成功！（无该元素则不删除）
请按任意键继续. . .

```

```

C:\ D:\00编程作业\Data Structure\T
20 62
请按任意键继续. . .

```

(5) 储存数据


```
C:\ D:\00编程作业\Data S
----int链表----
1. 显示数据--
2. 插入数据--
3. 删除数据--
4. 搜索数据--
5. 读取数据--
6. 储存数据--
0. 返回--
输入你的选择: 6_
```

```
C:\ D:\00编程作业\Data Structure\The Linked Lis
输入你要写入的文件名:intint01.txt
写入成功
请按任意键继续. . . _
```

(6) 读取数据 (退出程序再次进入)

```
C:\ D:\00编程作业\Data Str
----int链表----
1. 显示数据--
2. 插入数据--
3. 删除数据--
4. 搜索数据--
5. 读取数据--
6. 储存数据--
0. 返回--
输入你的选择: 5_
```

```
C:\ D:\00编程作业\Data Structure\The Linked Li
输入你要读取的文件名:intint01.txt
读取成功 (请确认文件里为int数据)
请按任意键继续. . . _
```

```
C:\ D:\00编程作业\Data
----int链表----
1. 显示数据--
2. 插入数据--
3. 删除数据--
4. 搜索数据--
5. 读取数据--
6. 储存数据--
0. 返回--
输入你的选择: 1_
```

```
C:\ D:\00编程作业\Data Structure\
20 62
请按任意键继续. . . _
```

2. Student 类型测试

```
C:\ D:\00编程作业\Data Structure\The Linked List\Debug\The Linked List.exe
选择链表的类型
1. int类型
2. Student类型
0. 退出
输入你的选择:2
```

(1) 插入数据:

```

D:\00编程作业\Data S
Stu链表-----
1. 显示数据-----
2. 插入数据-----
3. 删除数据-----
4. 搜索数据-----
5. 读取数据-----
6. 储存数据-----
0. 返回-----
输入你的选择: 2
Stu 插入数据
1. 在指定元素前插入数据
2. 在指定元素后插入数据
3. 在指定位置插入数据
4. 添加元素 (默认为最后位置)
0. 返回
输入你的选择: 4
输入你要添加的元素:
输入姓名:李四
输入ID:111
输入年龄:16
添加成功
请按任意键继续. . .

```

```

D:\00编程作业\Data Structure\The l
Stu 插入数据
1. 在指定元素前插入数据
2. 在指定元素后插入数据
3. 在指定位置插入数据
4. 添加元素 (默认为最后位置)
0. 返回
输入你的选择: 2
输入你想要插入后元素:
输入姓名:李四
输入ID:111
输入年龄:16
输入你想要插入元素:
输入姓名:张三
输入ID:222
输入年龄:22
插入成功!
请按任意键继续. . .

```

```

D:\00编程作业\Data Structure\The l
Stu 插入数据
1. 在指定元素前插入数据
2. 在指定元素后插入数据
3. 在指定位置插入数据
4. 添加元素 (默认为最后位置)
0. 返回
输入你的选择: 4
输入你要添加的元素:
输入姓名:王五
输入ID:333
输入年龄:33
添加成功
请按任意键继续. . .

```

(2) 显示数据

```

D:\00编程作业\Data
Stu链表-----
1. 显示数据-----
2. 插入数据-----
3. 删除数据-----
4. 搜索数据-----
5. 读取数据-----
6. 储存数据-----
0. 返回-----
输入你的选择: 1
一共有3名学生
姓名 学号 年龄
李四 111 16
张三 222 22
王五 333 33
请按任意键继续. . .

```

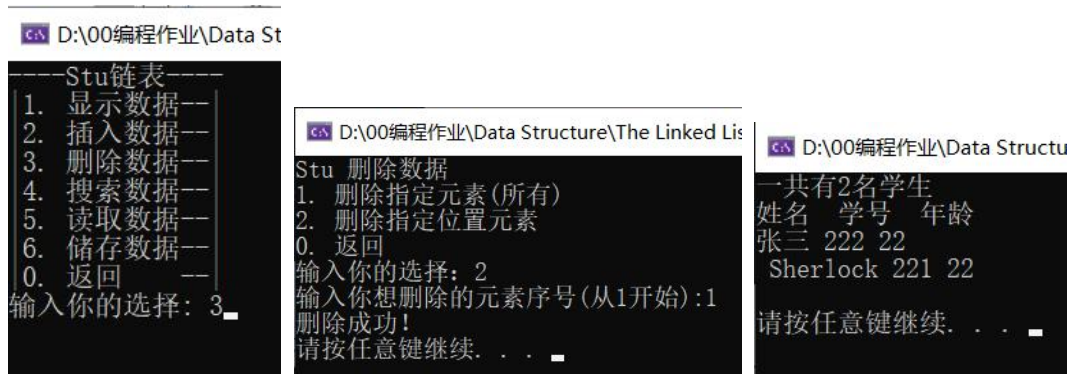
(3) 搜索数据

```

D:\00编程作业\Data Structure\
Stu 查询数据
1. 查找某元素位置
2. 修改指定位置元素
3. 将所有元素A改为B
0. 返回
输入你的选择: 2
输入你要修改的元素位置:2
输入你要修改为的元素:
输入姓名:Sherlock
输入ID:221
输入年龄:22
修改成功
请按任意键继续. . .
D:\00编程作业\Data S
Stu链表-----
1. 显示数据-----
2. 插入数据-----
3. 删除数据-----
4. 搜索数据-----
5. 读取数据-----
6. 储存数据-----
0. 返回-----
输入你的选择: 4
一共有3名学生
姓名 学号 年龄
李四 111 16
张三 222 22
Sherlock 221 22
请按任意键继续. . .

```

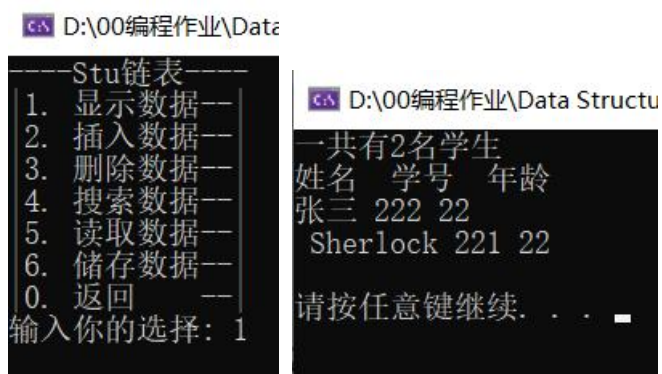
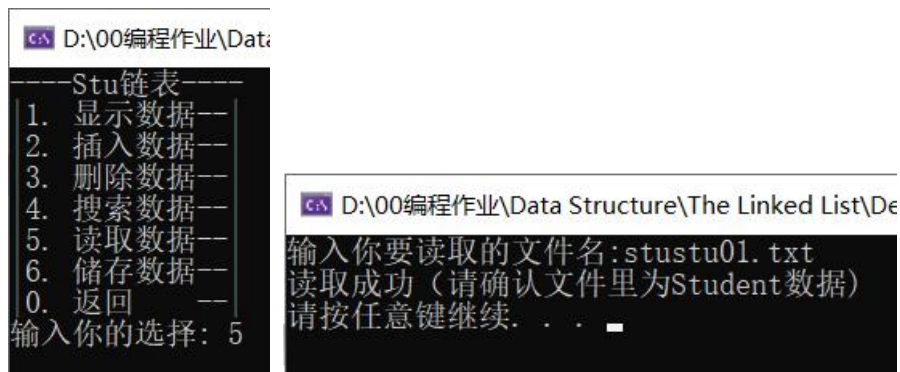
(4) 删除数据



(5) 储存数据



(6) 读取数据 (退出程序后再次打开)



八、实验分析

(1) 算法的性能分析

①插入元素算法 (Insert):

```
/// <summary>
/// 在最后添加元素
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="ele">要添加的元素</param>
template<class T>
void LinkedList<T>::PushBack(T ele)
{
    LinkNode<T>* n = new LinkNode<T>(ele);

    last->link = n;
    last = n;
    length++;
}
```

此 Insert 函数为在指定位置插入的方法。

Insert 函数要通过链节一步一步访问到该指定下标位置，算法复杂度为 $O(n)$ ，当元素极多时效率不高。

②添加元素算法 (PushBack):

```
/// <summary>
/// 在最后添加元素
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="ele">要添加的元素</param>
template<class T>
void LinkedList<T>::PushBack(T ele)
{
    LinkNode<T>* n = new LinkNode<T>(ele);

    last->link = n;
    last = n;
    length++;
}
```

PushBack 函数为在最后添加元素，通过尾结点直接定位最后一个节点，所以算法复杂度为 $O(1)$ ，效率高。

③删除元素算法 (Erase):

```
/// <summary>
/// 删除指定元素 (所有)
/// </summary>
```

```

/// <typeparam name="T"></typeparam>
/// <param name="ele">要删除的元素</param>
template<class T>
void LinkedList<T>::Erase(T ele)
{
    LinkNode<T>* nn = this->first;
    while (nn->link != NULL) {
        if (nn->link->data == ele) {
            LinkNode<T>* toDel = nn->link; //要删除的节点

            if (last == toDel)
                last = first;

            nn->link = toDel->link;
            delete toDel;
            length--;
            continue;
        }
        nn = nn->link;
    }
    MoveLast();
}

```

删除算法和插入算法类似，算法复杂度为 $O(n)$ ，当元素极多时效率低。

④搜索算法：

```

/// <summary>
/// 查找元素下标
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="ele">查找的元素</param>
/// <returns>元素下标(-1 为未找到)</returns>
template<class T>
int LinkedList<T>::Search(T ele)
{
    if (length == 0)
        return -1;

    LinkNode<T>* n = this->first->link;
    int pos = 0;
    while (n != NULL) {
        if (n->data == ele)
            return pos;
        n = n->link;
        pos++;
    }
}

```

```
    }  
    return -1;  
}
```

搜索算法也与插入算法类似,通过链节一步一步找元素,算法复杂度为 $O(n)$,当元素极多时算法效率低。

(2) 数据结构的分析

该链表插入、删除、搜索算法在数据极多时效率低,添加元素效率高。
适用于元素少的数据。

九、实验总结

此次实验提高了我对于指针的使用能力。开始写尾结点 (last) 时,耗费了大量时间也没完成,后面查询了资料,再次尝试终于完成 last 的设计,也就实现了 PushBack 函数。