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　曙光班

姓名　学号：周迎川　631907060434

实验项目名称：顺序表实验

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 16 日

教师评阅意见： 签名：　　年　月　日
实验成绩：

一、实验目的

验证实现的链表可适用于多种数据类型（简单数据类型和自定义数据类型，如学生类等）。熟悉并理解链表操作。

二、实验内容及要求

实现的顺序表可适用于多种数据类型（简单数据类型和自定义数据类型，如学生类等），并进行测试。

三、系统分析

（1）数据方面：

需要存储数据，并且数据个数不定，需要存储能存储的最大数量，并且可以进行随机插入，随机删除，随机查找。

（2）功能方面：

1、必须要能够扩展数据储存的内存，数据可能会很多，必须要能够动态分配；

2、需要又一个能知道链表长度的操作。

3、必须要有一个一个构造和析构操作。

4、必须要可以对数据元素的增加删除操作，那么也必须要有判断链表是否为空的操作。

5、需要一个能够获取链表中指定元素值的操作，类是具有封装性的，外部函数是不能访问的。

6、需要一个能遍历整个链表的操作。

7、由于数据量比较大，需要文件的读取和存储。

四、系统设计

（1）设计的主要思路

说明整体设计思路

根据题目要求：实现的顺序表可适用于多种数据类型（简单数据类型和自定义数据类型，如学生类等），必须要采用类模板才能实现，先设计一个链表，链表将各个结点定义为一个 struct 类，未来让外部可以访问，在定义一个链表类来存储头指针和尾部指针，定义为 private 类型，使其在类外不能被访问，以为链表头指针和尾部指针不能被访问，因此链表的结点也不能被外部访问。

然后再自定义一个学生类类型进行测试。

（2）数据结构的设计

数据结构设计思路

```
#ifndef LIST_H
#define LIST_H
#include <iostream>
using namespace std;
template <typename T>
struct ListNode
{
    T data;
    ListNode *next;
    ListNode(ListNode<T> *ptr=nullptr)//构造函数
    {
        next=ptr;
       // cout<<"Listnode created"<<endl;
    }
    ListNode(const T&data,ListNode<T> *ptr=nullptr)//构造函数
    {
        this->data=data;next=ptr;
        //cout<<"LIstnode created"<<endl;
    }
};
template <typename T>
class List
{
private:
    ListNode<T> *head,*last;
public:
    List();//构造函数
    List(const T&x);//构造函数
    ~List();//析构函数
    void makeEmpty();//将链表清空
    int Length();//获取链表长度
    ListNode<T> *getNextLast() const;//获取倒数第二个元素指针
    ListNode<T> *Search(const T&x);//搜索元素位置
```

```
    ListNode<T> *Locate(const int i);//获取第 i 个元素的位置
    int Find(const T&x);//找到 x 是第几个节点
    bool Insert(int i,const T &x);//在第 i 个位置插入元素
    bool Push_back(const T&x);//尾部插入元素
    bool Remove(const T &x);//删除指定元素
    bool Pop();//删除最后一个元素
    bool isEmpty();//判空
    bool RemovePos(int pos);//删除指定位置元素
    void Display();//显示整个表
    bool SaveFile(const char FileName[]);//保存文件
    bool ReadFile(const char FileName[]);//读取文件
};

#endif // LIST_H
```

1、先设计一个链表，链表将各个结点定义为一个 struct 类，未来让外部可以访问，在定义一个链表类来存储头指针和尾部指针，定义为 private 类型，使其在类外不能被访问，以为链表头指针和尾部指针不能被访问，因此链表的结点也不能被外部访问。

2、顺序表具有增加，删除，查找，遍历等基本功能。

3、由于数据类型可以是基本数据类型，也可以是自定义数据类型，所以用模板类实现。

自定义一个学生类测试程序。

学生类：

```
class Student
{
private:
    string id;
    string name;
public:
    Student()//构造函数
    {

    }
    Student (string i, string n)//含参构造函数
    {
        id=i;name=n;
    }
//    Student( const Student& s);
    ~Student()
    {
```

```
    }
    Student operator=(Student s);//重载赋值运算
    bool operator==(Student s);//重载==
    friend ostream& operator<<(ostream &cout, Student s);//重载<<
    friend istream& operator>>(istream &cin, Student &s);//重载>>
    void setIdName(string i,string n);//设置名字和学号
};
```

1、定义名字和学号两个基本数据。

2、重载一些运算符，=、==、<<、>>、这几个运算符，程序中必须要用到的运算符。

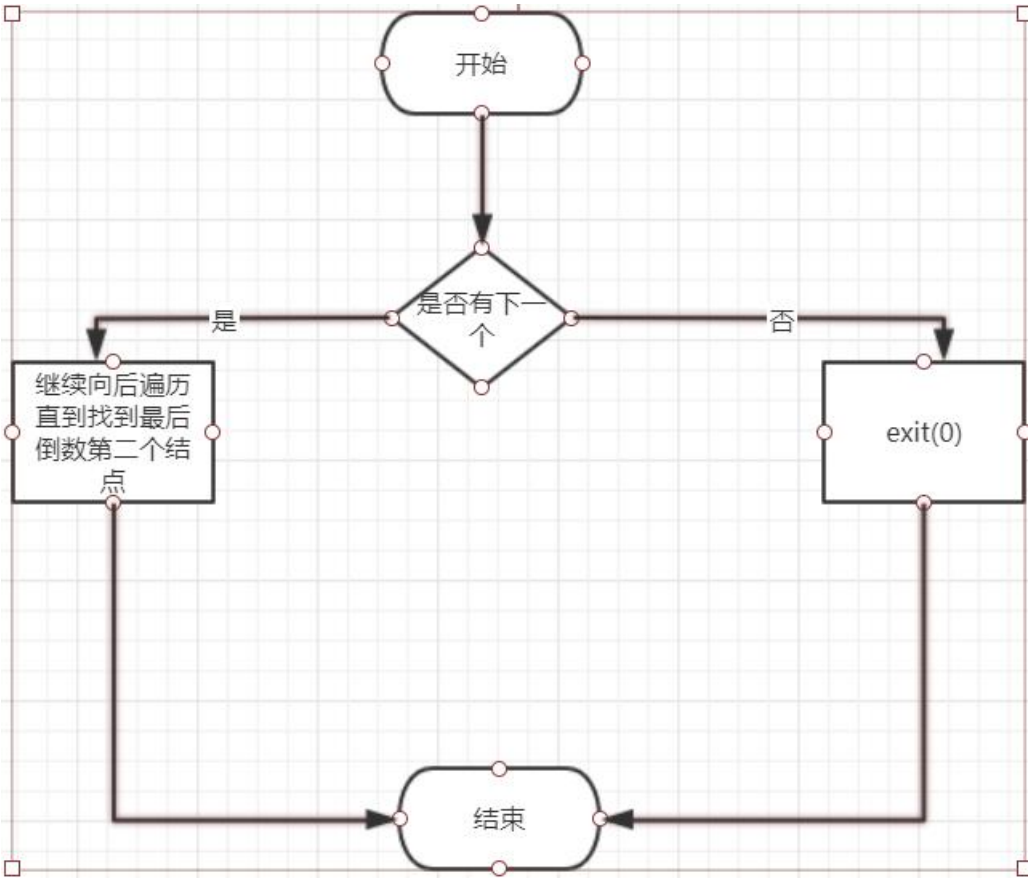
3、将<<和>>定义为友元函数，使其获得能够范围跟类中 private 的权限，以改变数据。

（3）基本操作的设计

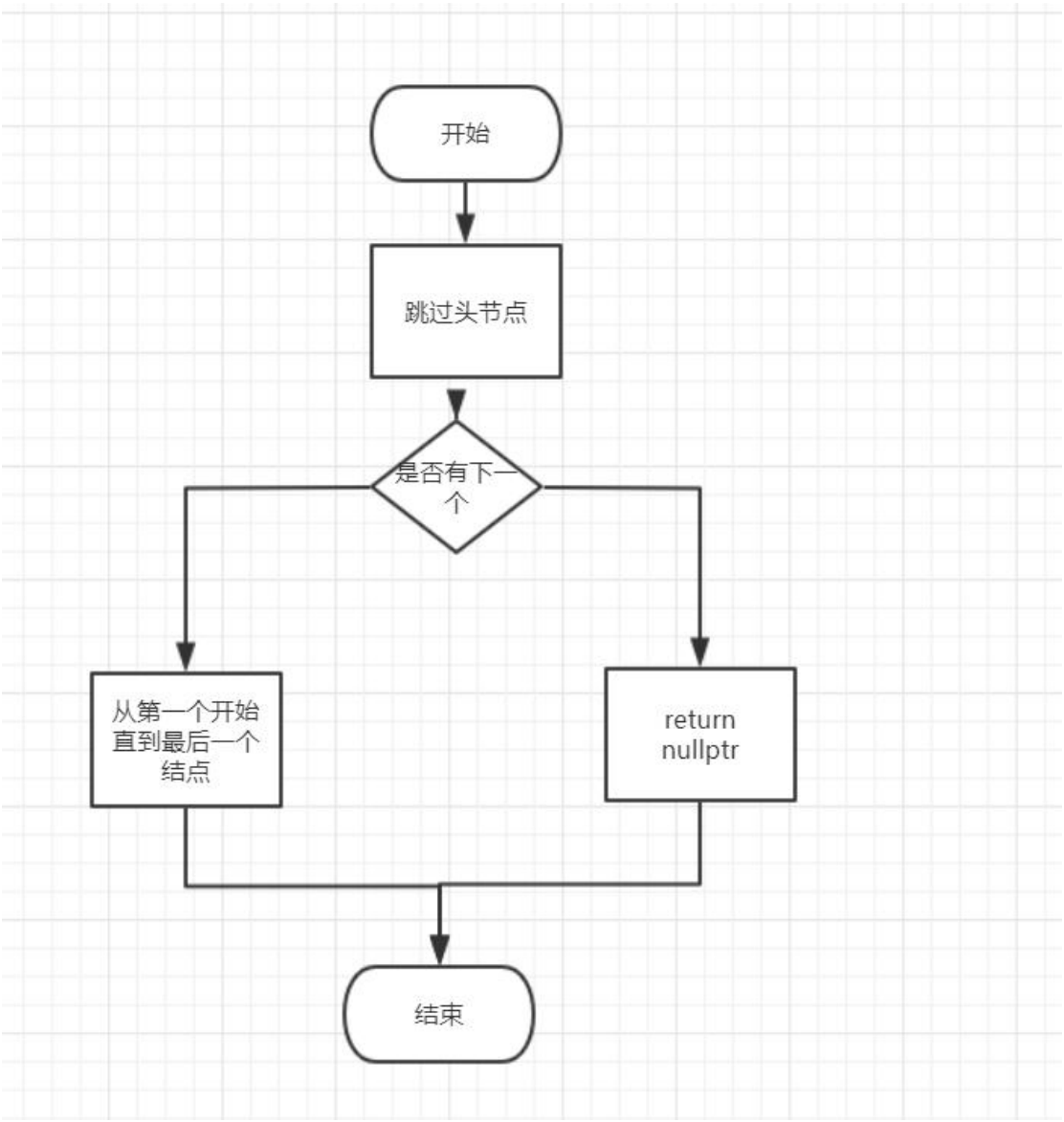
基本操作的抽象描述，关键算法的设计思路和算法流程图。

基本操作的抽象描述一般为操作名，初始条件，操作结构，参数说明等。

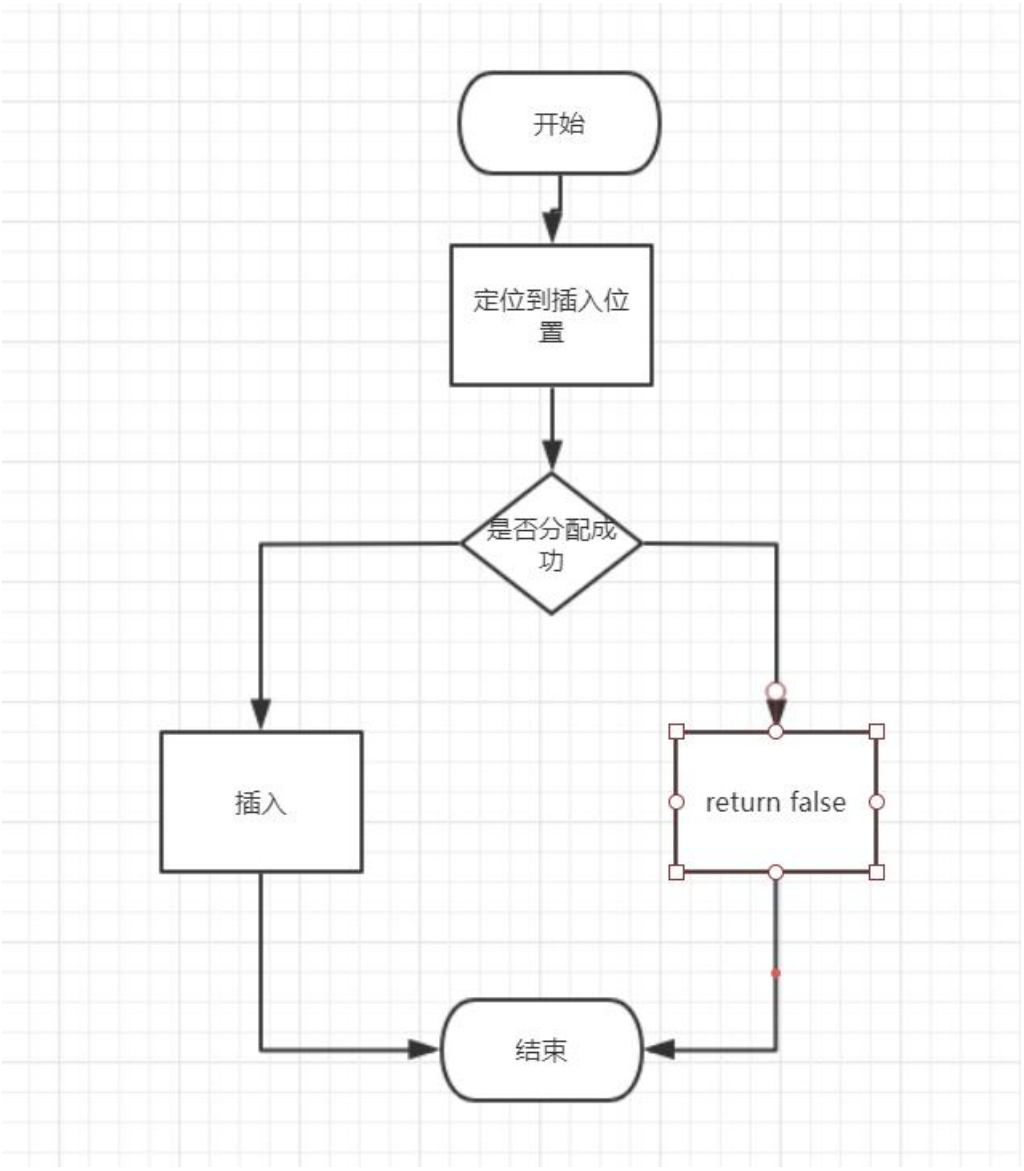
```
ListNode<T>* List<T>::getNextLast() const
//返回倒数第二个结点
```



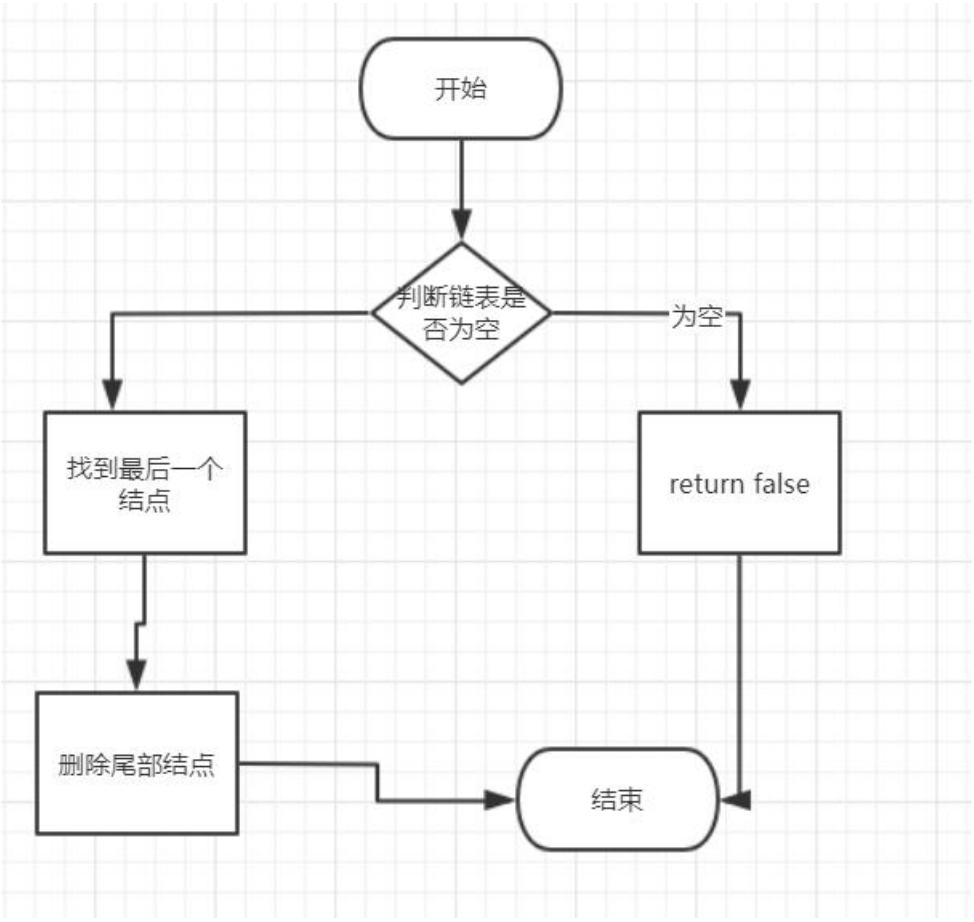
```
ListNode<T>* List<T>::Search(const T&x)
{//从头到尾一次寻找，如果寻找到，返回指针位置，如果没有找到返回空指针
```



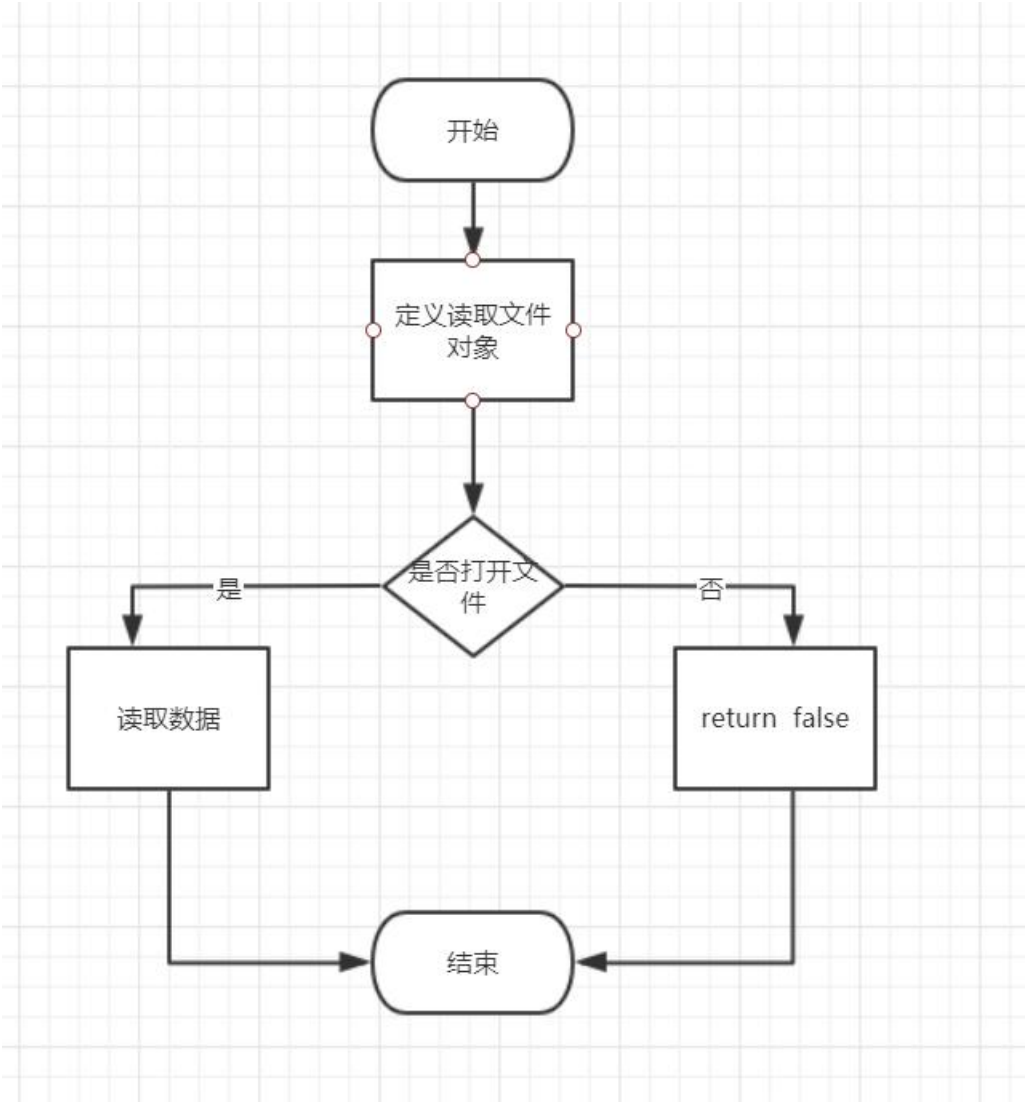

```
bool List<T>::Insert(int i,const T&x)
{
    //定位到 i-1 的位置，然后插入元素
```




```
bool List<T>::Pop()
{
    //删除最后一个结点，如果只有一个结点，删除该节点并将 last=head
    //如果有多个节点，定位到倒数第二个结点，然后删除最后一个，将 last 等于倒数第二个结点
```




```
template <typename T>
bool List<T>::ReadFile(const char FileName[])
{
    //首先定义读取文件的对象
     //打开文件，如果打开文件失败，提示打开失败
```



五、编程环境与实验步骤

（1）编程环境

主要是操作系统、编程工具软件

主要操作系统：Windows 10

编程工具：Qt

（2）实验步骤

只说明程序相关的各种文件创建步骤及文件的作用，不需说明文件的具体内容。

list.h　链表头文件

list.cpp　链表的实现文件

student.h　测试类，student 的头文件

student.cpp　　测试类 student 实现文件

main.cpp　　测试程序的主文件

SeqListdata.txt　　int 类数据文件

SeqListStudent.txt　　Student 类数据文件

（3）编译参数

若有特殊的编译参数设置，需说明详细步骤。

若无特殊的编译参数设置，则只需简单说明操作步骤。

六、实现代码

主要功能的实现代码

```
#include "list.h"
#include <iostream>
#include <cstdlib>
#include <fstream>
using namespace std;
template <typename T>
List<T>::List()
{
    head=new ListNode<T>;
    last=head;
    head->next=nullptr;
    //cout<<"List crated"<<endl;
}
template <typename T>
List<T>::List(const T&x)
{
    head=new ListNode<T>(x);
    last=head;
    head=nullptr;
    //cout<<"List crated"<<endl;
}
template <typename T>
List<T>::~List()
{
    makeEmpty();
}
template <typename T>
void List<T>::makeEmpty()
{
    //摧毁链表
```

```
    ListNode<T> *now;
    while(head->next!=nullptr)
    {
        now=head->next;
        head->next=now->next;
        delete now;
    }
}
template <typename T>
int List<T>::Length()
{
    int account=0;
    ListNode<T> *p=head->next;//从头节点后面一个开始计数
    while(p!=nullptr)
    {
        p=p->next;
        account++;
    }
    return account;
}
template <typename T>
ListNode<T>* List<T>::getNextLast() const
{//返回倒数第二个结点
    ListNode<T> *now = head->next;
    if(now->next==last)
        return nullptr;
    while(now->next!=last)
        now=now->next;
    return now;

}
template <typename T>
ListNode<T>* List<T>::Search(const T&x)
{//从头到尾一次寻找，如果寻找到，返回指针位置，如果没有找到返回空指针
    ListNode<T> *now=head->next;//跳过头节点
    while(now->next!=nullptr)
    {
        now=now->next;
        if(now->data==x)
            return now;
    }
    return nullptr;
}
template <typename T>
```

```
ListNode<T>* List<T>::Locate(int i)
{
    //定位第 i 个元素位置，如果总共没有 i 个，那么返回 now=nullptr；如果找到了第 i 个位置，返回地址值
    ListNode<T> *now=head;
    int k=0;
    while(now->next!=nullptr&&k<i)
    {
        k++;now=now->next;
    }
    return now;
}
template <typename T>
int List<T>::Find(const T&x)
{
    {//从头到尾一次寻找，如果寻找到，返回是第几个，如果没有找到返回-1
        ListNode<T> *now=head->next;//跳过头节点
        int num=1;
        while(now->next!=nullptr)
        {
            num++;
            now=now->next;
            if(now->data==x)
                return num;
        }
        return -1;
    }
}
template <typename T>
bool List<T>::Insert(int i,const T&x)
{
    //定位到 i-1 的位置，然后插入元素
    ListNode<T>* now=Locate(i-1);
    ListNode<T>* in=new ListNode<T>(x);
    if(in==nullptr)
    {
        cerr<<"插入失败！！！"<<endl;
        return false;
    }
    in->next=now->next;
    now->next=in;
    return true;
}
template <typename T>
```

```
bool List<T>::Push_back(const T&x)
{//在尾部插入节点，需要尾部指针指向差人元素，next 空间指向 NULL
    ListNode<T>* in=new ListNode<T>(x);
    //cout<<"3"<<endl;
    if(in==nullptr)
    {
        cerr<<"插入失败！！！"<<endl;
        return false;
    }
    in->next=last;
    last->next=in;
    last=in;
    last->next=nullptr;
    //cout<<"pushover"<<endl;
    return true;
}
template <typename T>
bool List<T>::Remove(const T&x)
{
    //找到改元素所在的位置，找到改元素前一个节点位置和该元素位置
    if(isEmpty())
    {
        cerr<<"链表为空！！！"<<endl;
        return false;
    }
    int i=Find(x);
    ListNode<T> *now=Locate(i-1);
    ListNode<T> *del=Locate(i);
    now->next=del->next;
    delete del;
    if(last==del)//如果删除的结点是最后一个结点
    {
        delete del;
        last=now;
        return true;
    }
    return true;
}
template <typename T>
bool List<T>::Pop()
{
    //删除最后一个结点，如果只有一个结点，删除该节点并将 last=head
    //如果有多个节点，定位到倒数第二个结点，然后删除最后一个，将 last
等于倒数第二个结点
```

```
    ListNode<T> *now=head->next;
    ListNode<T> *del;
    if(isEmpty())
    {
        cerr<<"链表为空！！！"<<endl;
        return false;
    }
    if(now->next==nullptr)
    {
        del=last;
        last=head;
        delete del;
        return true;
    }
   now=getNextLast();
   del=last;
   now->next=nullptr;
   delete del;
   last=now;
   last->next=nullptr;
   return true;
}
template <typename T>
bool List<T>::isEmpty()
{
    if(head->next==nullptr)
    {
        cerr<<"链表为空！！！"<<endl;
        return true;
    }
    else
        return false;
}
template <typename T>
bool List<T>::RemovePos(int i)
{
    //定位到 i-1 的位置，然后删除元素
    //找到改元素所在的位置，找到改元素前一个节点位置和该元素位置
    if(isEmpty())
    {
        cerr<<"链表为空！！！"<<endl;
        return false;
    }
    ListNode<T> *now=Locate(i-1);
```

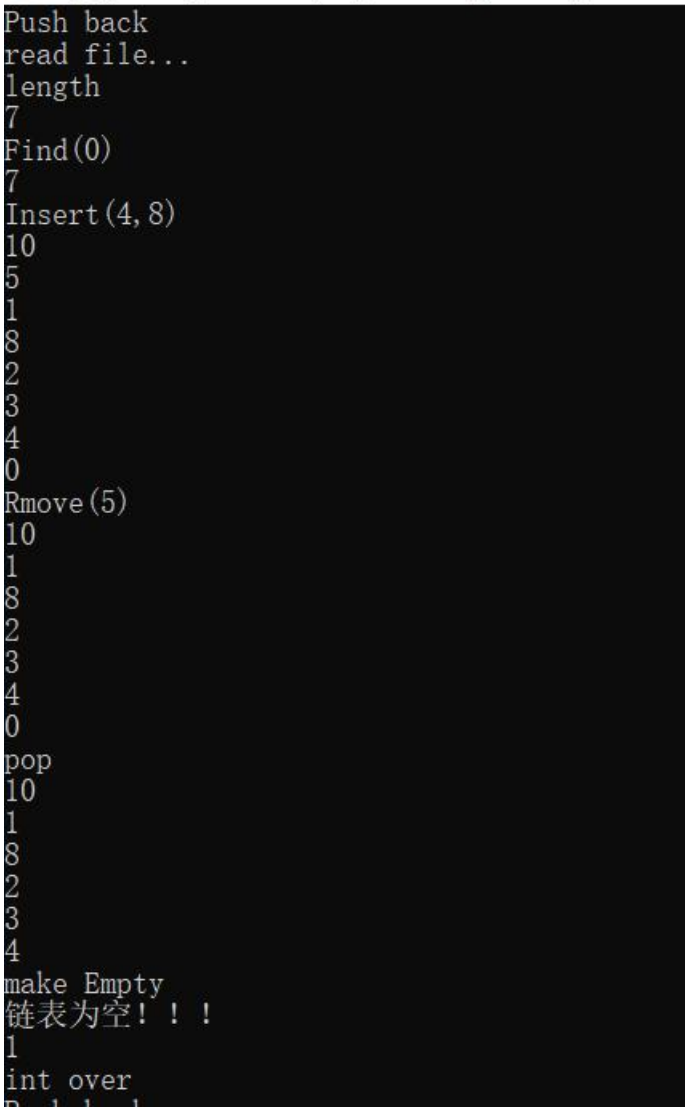
```
    ListNode<T> *del=Locate(i);
    now->next=del->next;
    delete del;
    if(last==del)//如果删除的结点是最后一个结点
    {
        delete del;
        last=now;
        return true;
    }
    return true;
}
template <typename T>
void List<T>::Display()
{//跳过头节点，遍历
    ListNode<T>* now=head->next;
    //cout<<"2"<<endl;
    while(now!=nullptr)
    {
        cout<<now->data<<endl;
        now=now->next;
    }
}
template <typename T>
bool List<T>::SaveFile(const char FileName[])
{
    //首先定义存文件的对象
     //打开文件，如果打开文件失败，提示打开失败
    ListNode<T> *now=head->next;
    ofstream fout;
    fout.open(FileName);
    if(fout.fail())
    {
        cerr<<"打开文件失败！！！"<<endl;
        return false;
    }
    while(now->next!=nullptr)
        fout<<now->data<<endl;
    return true;
}
template <typename T>
bool List<T>::ReadFile(const char FileName[])
{
    T in;
    ifstream fin;
```

```
    fin.open(FileName);
    if(fin.fail())
    {
        cerr<<"打开文件失败！！！"<<endl;
        exit(1);
    }
    while (fin>>in)
    {
        Push_back(in);
       // cout<<"gogogo"<<endl;
    }
    return true;
}
```

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。

Int 型测试



Student 类型测试

```
Push back
read file...
length
6
Find(s1)
2
Insert(4,s2)
          66666                  wang
   631907060434          zhouyingchuan
   631907060520               zhangsan
   630707060434                   dong
   631707060520              quhongyang
   631907060305           dongjiachneg
   631907060416                fangmin
Rmove(s1)
          66666                  wang
   631907060520               zhangsan
   630707060434                   dong
   631707060520              quhongyang
   631907060305           dongjiachneg
   631907060416                fangmin
pop
          66666                  wang
   631907060520               zhangsan
   630707060434                   dong
   631707060520              quhongyang
   631907060305           dongjiachneg
make Empty
链表为空！！！
1
student over
```

八、实验分析

（1）算法的性能分析

主要针对增加、删除、搜索等算法。

在尾部增加算法的时间复杂度为 O(n)

在指定位置插入元素的时间复杂度为 O((n+1)/2)

删除指定元素的时间复杂度为 O((n+1)/2)

删除指定位置的元素时间复杂度为 O((n+1)/2)

搜索算法的时间复杂度为 O((n+1)/2)

（2）数据结构的分析

通过性能分析总结此种存储结构的优缺点，并说明其适用场景。

优点：可以任意添加元素，在任意位置，可以任意删除任意位置的元素，而且效率高，可以灵活添加元素。

缺点：占用空间比普通数组大，不能随机访问元素，如果元素太多，则访问需要较长的时间。

使用场景：适用于任何数据，而且可以快速灵活增加删除，便于管理。

九、实验总结

主要针对本实验的分析、设计、实现、测试等环节进行总结，包含收获与不足，此部分的阐述应较为详细。

本实验需要用多种数据类型测试，所以需要用模板。设计的时候，先设计了一个结点类和一个链表类，然后再设计了一个学生类进行测试。实现的时候，具体的操作据说再增加删除，搜索需要的时候再去写操作。测试的时候用文件测试，用了多组数据测试，用学生类文件和整型数据测试的。在写链表的时候，要先建立一个结点类，让结点类来存储地址和数据，再建立一个链表类，再类中定义头部指针和尾部指针，方便链表的操作。

此次实验让我有进一步熟悉了一下 C++语法，之前很多语法已经忘记了，现在又想了起来。由于对 C++的部分东西已经遗忘，写程序的时候看来很多以前的课件。文件操作不能读中文，我感觉是因为文件格式是 utf-8，而我写代码通常编码是 gb2312，所以读文件会乱码。以后我可能会更多的选择 utf-8 了，这个编码使用的范围更广。

附录

参考文献：

1.
2.
3.