# 实验二：单链表

#### （一）实验题目

题目一**：尾插法创建单链表，打印创建结果。**

题目二：**头插法创建单链表，打印创建结果。**

题目三：**销毁单链表。**

题目四：**求链表长度。**

题目五：**求单链表中第i个元素（函数），若不存在，报错。**

* + **第一组数据：单链表长度n≥10，i分别为5，n，0，n+1，n+2**
  + **第二组数据：单链表长度n=0，i分别为0，2**

题目六：**在第i个结点前插入值为x的结点。**

* + **第一组数据：单链表长度n≥10，x=100,**

**i分别为5,n,n+1,0,1,n+2**

* + **第二组数据：单链表长度n=0，x=100，i=5**

题目七：**链表中查找元素值为x的结点，成功返回结点指针，失败报错。**

* + **单链表元素为 （1，3，6，10，15，16，17，18，19，20）**

**x=1，17，20，88**

题目八**：删除单链表中第i个元素结点。**

* + **第一组数据：单链表长度n≥10，i分别为5,n,1,n+1,0**
  + **第二组数据：单链表长度n=0， i=5**

题目九**：在一个递增有序的单链表L中插入一个值为x的元素，并保持其递增有序特性。**

* + **单链表元素为 （10,20,30,40,50,60,70,80,90,100）,**

**x分别为25，85，110和8**

题目十**：将单链表Ｌ中的奇数项和偶数项结点分解开（元素值为奇数、偶数），分别放入新的单链表中，然后原表和新表元素同时输出到屏幕上，以便对照求解结果。**

* **第一组数据：单链表元素为**

**（1,2,3,4,5,6,7,8,9,10,20,30,40,50,60）**

* + **第二组数据：单链表元素为 （10,20,30,40,50,60,70,80,90,100）**

题目十一：**求两个递增有序单链表L1和L2中的公共元素，放入新的单链表L3中。**

* **第一组**
  + **第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）**
  + **第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）**
* **第二组**
  + **第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）**
  + **第二个单链表元素为 （2，4，5，7，8，9，12，22）**
* **第三组**
  + **第一个单链表元素为 （）**
  + **第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10）**

题目十二：**删除递增有序单链表中的重复元素，要求时间性能最好。**

* + **第一组数据：单链表元素为 （1,2,3,4,5,6,7,8,9）**
  + **第二组数据：单链表元素为**

**（1,1,2,2,2,3,4,5,5,5,6,6,7,7,8,8,9）**

* + **第三组数据：单链表元素为 （1,2,3,4,5,5,6,7,8,8,9,9,9,9,9）**

题目十三：**递增有序单链表L1、L2，不申请新结点，利用原表结点对两表进行合并，并使得合并后成为一个集合，合并后用L1的头结点作为头结点，删除L2的头结点，要求时间性能最好。**

* **第一组**
  + **第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）**
  + **第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）**
* **第二组**
  + **第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）**
  + **第二个单链表元素为 （2，4，5，7，8，9，12，22）**
* **第三组**
  + **第一个单链表元素为 （）**
  + **第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10）**

**\* 单链表扩展实验**

* + **非必做内容，有兴趣的同学选做，**

**<1>（递增有序）单链表表示集合A、B，实现：**

* + **C=A**∩**B，C=A**∪**B，C=A-B**
  + **A=A**∩**B，A=A**∪**B，A=A-B**

**<2>（递增有序）单链表表示集合A、B，判定A是否B的子集。**

**<3>(2009)（15分）已知一个带有表头结点的单链表，结点结构如下图。假设该链表只给出了头指针list。在不改变链表的前提下，请设计一个尽可能高效的算法，查找链表中倒数第k个位置上的结点（k为正整数）。若查找成功，算法输出该结点的data值，并返回1；否则，只返回0。要求：**

**（1）描述算法的基本设计思想**

**（2）描述算法的详细实现步骤**

**（3）根据设计思想和实现步骤，采用程序设计语言描述算法（使用C 或C++语言实现），关键之处请给出简要注释。**

**<4>(2011)（15 分）一个长度为L（L≥1）的升序序列S，处在第 个位置的数称为S 的中位数。例如，若序列S1=(11, 13, 15, 17, 19)，则S1 的中位数是15。两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若S2=(2, 4, 6, 8, 20)，则S1 和S2 的中位数是11。**

**现有两个等长升序序列A 和B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列A 和B 的中位数。要求：**

**（1）给出算法的基本设计思想。**

**（2）根据设计思想，采用C 或C++语言描述算法，关键之处给出注释。**

**（3）说明你所设计算法的时间复杂度和空间复杂度。**

#### （二） 实验源码

#pragma once

#include <iostream>

using namespace *std*;

typedef struct node

{

int data;

node\* next;

}List;

class linkedList

{

public:

linkedList();//构造函数

~linkedList();//析构函数，<4>销毁单链表

void reList();//初始化单链表

void tailList(int l);//<1>尾插法构建单链表

void headList(int l);//,<2>头插法构建单链表

int ListLength();//<4>单链表长度

void getMem(int i);//<5>求单链表中第i个元素（函数），若不存在，报错

void insertx(int i, int x);//<6>在第i个结点前插入值为x的结点

void addMem(int x); //警告：必须初始化才能使用！

bool searchMem( int &i,int x);//<7>链表中查找元素值为x的结点，成功返回结点指针，失败报错。

bool isEmpty();//判断单链表是否为空

void deleteMem(int i);//<8>删除单链表中第i个元素结点

void insertList(int value);//<9>在一个递增有序的单链表L中插入一个值为x的元素，并保持其递增有序特性

void sortList(linkedList& A, linkedList& B);//将调用单链表中的元素按奇偶性分配給被调用的单链表LA与LB

void AjiaoB(linkedList& A, linkedList& B, linkedList& C);

void delsame(linkedList&A);

void AbingB(linkedList&A,linkedList&B);

void printList();//打印单链表

private:

List\* head;

int length;

};

#include "list.h"

linkedList::linkedList()

{

head = *NULL*;

length = 0;

}

linkedList::~linkedList()

{

List\* tmp = head;

while (tmp->next)

{

List\* u= tmp;

tmp = tmp->next;

delete u;

}

}

void linkedList::reList()

{

head = new List;

if (!head)

{

*cout* << "链表初始化失败" << *endl*;

}

head->next = *NULL*;

*cout* << "链表初始化成功" << *endl*;

}

void linkedList::tailList(int l)

{

reList();

List\* tail = head;

*cout* << "请输入"<<l<<"个元素" << *endl*;

for (int i = 1; i <= l; i++)

{

List\* tmp = new List;

int num;

*cin* >> num;

tmp->data = num;

tmp->next = tail->*next*;

tail->*next* = tmp;

tail = tmp;

length++;

}

*cout* << "尾插法创建单链表成功" << *endl*;

}

void linkedList::headList(int l)

{

reList();

*cout* << "请输入" << l << "个元素" << *endl*;

for (int i = 0; i < l; i++)

{

int num;

*cin* >> num;

List\* tmp = new List;

tmp->data = num;

tmp->next = head->next;

head->next = tmp;

length++;

}

*cout* << "头插法创建单链表成功" << *endl*;

}

/\*

单链表长度

\*/

int linkedList::ListLength()

{

if (isEmpty())

{

*cout* << "空链表！" << *endl*;

return 0;

}

int l = 0;

List\* tmp = head->next;

while (tmp)

{

tmp = tmp->next;

l++;

}

return l;

}

/\*/

求单链表中第i个元素（函数），若不存在，报错。

\*/

void linkedList::getMem(int i)

{

if (isEmpty())

{

*cout* << "链表为空获取元素失败" << *endl*;

}

else if ( i> length)

{

*cout* << "位置大于表长获取元素失败" << *endl*;

}

else if (i <= 0)

{

*cout* << "位置必须大于0获取元素失败" << *endl*;

}

else

{

int index = 0, num = 0;

List\* tmp = head;

while (tmp)

{

if (index == i)

{

num = tmp->data;

}

tmp = tmp->next;

index++;

}

*cout* << "第" << i << "个元素为" << num << *endl*;

}

}

/\*插入元素\*/

void linkedList::addMem(int x)

{

List\* tmp = head;

List\* last = *NULL*;

while (tmp)

{

last = tmp;

tmp = tmp->next;

}

List\* p = new List;

p->*data* = x;

p->*next* = *NULL*;

last->*next* =p;

length++;

}

/\*节点插入元素\*/

void linkedList::insertx(int i, int x)

{

if (isEmpty())

{

*cout* << "链表为空插入失败" << *endl*;

}

else if (i > length)

{

*cout* << "插入位置大于表长" << *endl*;

}

else if (i == length)

{

addMem(x);

}

else if (i <= 0)

{

*cout* << "位置必须大于0插入失败" << *endl*;

}

int index = 0;

List\* tmp = head;

while (index != i- 1 && tmp)

{

index++;

tmp = tmp->next;

}

if (tmp == *NULL*)

{

*cout* << "位置大于表长插入失败" << *endl*;

}

List\* p = new List;

p->*data* = x;

p->*next* = tmp->next;

tmp->next = p;

length++;

}

/\*

按值查找元素节点

\*/

bool linkedList::searchMem( int &i,int x)

{

if (isEmpty())

{

*cout* << "链表为空获取元素失败" << *endl*;

return false;

}

int time = 1;

List\* tmp = head->next;

while (tmp)

{

if (tmp->data == x)

{

i= time;

return true;

}

tmp = tmp->next;

time++;

}

return false;

}

/\*判断是否为空\*/

bool linkedList::isEmpty()

{

return head->next == *NULL*;

}

/\*删除链表元素\*/

void linkedList::deleteMem(int i)

{

if (isEmpty())

{

*cout* << "链表为空删除元素失败" << *endl*;

}

else if (i > length)

{

*cout* << "位置大于表长删除元素失败" << *endl*;

}

else if (i<= 0)

{

*cout* << "位置必须大于0删除元素失败" << *endl*;

}

else

{

List\* tmp = head;

int index = 0, x = 0;

while (index != i - 1 && tmp)

{

tmp = tmp->next;

index++;

}

List\* p = tmp->next;

x = p->*data*;

tmp->next = tmp->next->*next*;

delete p;

length--;

*cout* << "已删除第" << i << "个节点的元素" << x << *endl*;

}

}

/\*链表中按顺序插入元素\*/

void linkedList::insertList(int x)

{

if (isEmpty())

{

*cout* << "链表为空插入失败！" << *endl*;

}

List\* tmp = head;

while (tmp->next && tmp->next->*data* < x)//下一个节点的data比x小就继续循环

{

tmp = tmp->next;

}

List\* p = new List;

p->*data* = x;

p->*next* = tmp->next;

tmp->next =p;

}

/\*将原链表奇数偶数分开到两个链表\*/

void linkedList::sortList(linkedList& LA, linkedList& LB)

{

if (isEmpty())

{

*cout* << "原链表为空分离失败" << *endl*;

}

if (!LA.head && !LB.head)

{

LA.reList();

LB.reList();

}

List\* tmp = head->next;

while (tmp)

{

if (tmp->data %2==0)

LA.addMem(tmp->data);

else

LB.addMem(tmp->data);

tmp = tmp->next;

}

}

/\*将两个链表相同元素放到新链表3\*/

void linkedList::AjiaoB(linkedList& A, linkedList& B, linkedList& C)

{

int i = 0;

{

List\* pA,\* pB,\*pC;

pA = A.head->next;

pB = B.head->next;

pC = C.head->next;

while (pA != *NULL* && pB != *NULL*)

{

if (pA->*data* < pB->*data*) //a 小于 b

{

pA = pA->*next*; //后移pA，继续比较

}

else if (pA->*data*> pB->*data*) //大于

{

pB = pB->*next*; //就后移pB

}

else //相等，就尾插到C，然后同时后移pA，pB

{

pC = new List;

C.addMem(pA->*data*);

//尾插到C完毕，然后后移A，B

pA = pA->*next*;

pB = pB->*next*;

i++;

//cout << "\*\*" << i << "\*\*\*\*\*\*" << pC->data << endl;

}

}

pC->*next* = *NULL*;

}

C.printList();

}

/\*

删除重复元素

\*/

void linkedList::delsame(linkedList &A)

{

List\* p1, \* p2;

p1 = A.head;

p2 = p1->*next*;

while (p2 != *NULL*)

{

if (p1->*data*!= p2->*data*) //如果相邻两个结点元素不等

{

p1 = p1->*next*; //就同时后移p1,p2

p2 = p2->*next*;

}

else //相等，则后移p2直到第一个不等于p1的结点，

{

while (p2 != *NULL* && p2->*data* == p1->*data*)

{

p2 = p2->*next*;

}

p1->*next* = p2;

}

}

*cout* << "去除重复元素后";

A.printList();

}

/\*将 链表2合并到链表1为一个集合\*/

void linkedList::AbingB(linkedList& A, linkedList& B)

{

List\* p1, \* p2, \* t, \* s,\*u;

p1 = A.head->next; //从两个头结点开始遍历

p2 =B.head->next;

s = A.head;

while (p1 != *NULL* && p2 != *NULL*)

{

if (p1->*data* == p2->*data*) //如果表1当前元素等于表2的

{

t = p2;

p2 = p2->*next*;//后移p2，然后继续比较

delete t;

}

else if (p1->*data* <p2->*data*) //

{

s = p1;

p1 = p1->*next*; //就后移p1，并保证s始终是p1的前驱结点

}

else //大于，说明可以并入表1

{

u = p2->*next*;

p2 ->*next*= s->*next*;//表2插表1

s->*next* = p2;

s = s->*next*;

p2 = u;

}

}

while (p2 != *NULL*)//表2剩下的结点都能并入1

{

s->*next* = p2;

s = p2;

p2 = p2->*next*;

}

delete B.head;//释放掉L2头结点

B.head = *NULL*;

*cout* << "执行操作后";

A.printList();

}

/\*打印链表\*/

void linkedList::printList()

{

if (isEmpty())

{

*cout* << "空链表，无法打印！" << *endl*;

}

List\* tmp = head->next;

*cout* << "打印链表";

while (tmp)

{

*cout* << tmp->data << " ";

tmp = tmp->next;

}

*cout* << *endl*;

}

#include <iostream>

#include "list.h"

using namespace *std*;

int main()

{

/\*测试1234\*/

linkedList L1,L2,L3;

L1.tailList(5);

L1.printList();

*cout*<<"链表长度"<< L1.ListLength()<<*endl*;

L2.headList(5);

L2.printList();

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

/\*测试5\*/

linkedList tst51,tst52;

*cout* << "5>求单链表中第i个元素（函数），若不存在，报错," << *endl*

<< "第一组数据：单链表长度n≥10，i分别为5，n，0，n+1，n+2" << *endl*;

tst51.tailList(11);

tst51.getMem(5); tst51.getMem(11); tst51.getMem(0); tst51.getMem(12); tst51.deleteMem(1); tst51.getMem(13);

*cout*<<"第二组数据：单链表长度n = 0，i分别为0，2" << *endl*;

tst52.tailList(0);

tst52.getMem(0); tst52.getMem(5);

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

/\*测试8\*/

*cout* << "删除单链表中第i个元素结点" << *endl*

<< "单链表长度n=11，i分别为5, n, 1, n + 1, 0" << *endl*;

tst51.deleteMem(5); tst51.deleteMem(11); tst51.deleteMem(12); tst51.deleteMem(0);

*cout*<< "第二组数据：单链表长度n = 0， i = 5" << *endl*;

tst52.deleteMem(5);

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

/\*测试6\*/

*cout* << "在第i个结点前插入值为x的结点," << *endl*

<< "第一组数据：单链表长度n=11，x=100, i分别为5,n,n+1,0,1,n+2" << *endl*;

*cout* << "第二组数据：单链表长度n = 0，x = 100，i = 5" << *endl*;

linkedList tst61, tst62;

tst61.tailList(11);

tst61.insertx(5, 100);

*cout* << "插入后"; tst61.printList();

tst61.insertx(11, 100); *cout* << "插入后"; tst61.printList();

tst61.insertx(12, 100); *cout* << "插入后"; tst61.printList();

tst61.insertx(0, 100); *cout* << "插入后"; tst61.printList();

tst61.insertx(1, 100); *cout* << "插入后"; tst61.printList();

tst61.insertx(13, 100); *cout* << "插入后"; tst61.printList();

tst62.tailList(0); *cout* << "插入后"; tst62.printList();

tst62.insertx(5, 100); *cout* << "插入后"; tst62.printList();

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

/\*测试7\*/

*cout* << "链表中查找元素值为x的结点，成功返回结点指针，失败报错。" << *endl*;

*cout* << "单链表元素为 （1，3，6，10，15，16，17，18，19，20）" << *endl*;

*cout*<<"x = 1，17，20，88" << *endl*;

linkedList tst71, tst72;

tst71.tailList(10);

int p1, p2, p3, p4;

tst71.searchMem(p1, 1); tst71.searchMem(p2, 17); tst71.searchMem(p3, 20); tst71.searchMem(p4, 88);

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

/\*测试9\*/

*cout* << "在一个递增有序的单链表L中插入一个值为x的元素，并保持其递增有序特性。" << *endl*;

*cout* << "单链表元素为 （10, 20, 30, 40, 50, 60, 70, 80, 90, 100）" << *endl*;

*cout* << "x分别为25，85，110和8" << *endl*;

linkedList tst91;

tst91.tailList(10);

tst91.insertList(25); tst91.printList();

tst91.insertList(85); tst91.printList();

tst91.insertList(110); tst91.printList();

tst91.insertList(8); tst91.printList();

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

/\*测试10\*/

*cout* << "将单链表Ｌ中的奇数项和偶数项结点分解开（元素值为奇数、偶数)" << *endl*;

*cout* << "分别放入新的单链表中，然后原表和新表元素同时输出到屏幕上，以便对照求解结果。" << *endl*;

*cout* << "第一组数据：单链表元素为 （1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60）" << *endl*;

*cout* << "第二组数据：单链表元素为 （10, 20, 30, 40, 50, 60, 70, 80, 90, 100）" << *endl*;

linkedList tst101, tst102, tstLA, tstLB, tstLC,tstLD;

tst101.tailList(15);

tst101.sortList(tstLA,tstLB );

tst101.printList();

*cout* <<"偶数项" ; tstLA.printList();

*cout* <<"奇数项"; tstLB.printList();

tst102.tailList(10);

tst102.sortList(tstLC, tstLD);

tst102.printList();

*cout* <<"偶数项" ; tstLC.printList();

*cout* << "奇数项"; tstLD.printList();

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

/\*测试11\*/

/\*注：同时测试时由于第二组数据无公共元素所以无输出，

第三组元素由于第一个链表为空会终止程序

暂时只能分开测试

\*/

*cout* << "求两个递增有序单链表L1和L2中的公共元素，放入新的单链表L3中" << *endl*;

*cout* << "第一组" << *endl*;

*cout* << "第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）" << *endl*;

*cout* << "第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）" << *endl*;

*cout*<<"第二组" << *endl*;

*cout* <<"第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）" << *endl*;

*cout* <<"第二个单链表元素为 （2，4，5，7，8，9，12，22）" << *endl*;

*cout* <<"第三组 "<< *endl*;

*cout* << "第一个单链表元素为 （）" << *endl*;

*cout* << "第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10）" << *endl*;

*cout*<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<*endl*;

linkedList tst111, tst112, tst113,tst114,tst115,tst116,tst117,tst118,tst119;

tst111.tailList(10);

tst112.tailList(13);

tst113.reList();

tst113.AjiaoB(tst111, tst112,tst113);

tst114.tailList(10);

tst115.tailList(8);

tst116.reList();

tst116.AjiaoB(tst115, tst114, tst116);

tst117.tailList(0);

tst118.tailList(10);

tst119.reList();

tst119.AjiaoB(tst117, tst118, tst119);

*cout* << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

/\*测试12\*/

*cout* << "删除递增有序单链表中的重复元素，要求时间性能最好" << *endl*;

*cout* << "第一组数据：单链表元素为 （1,2,3,4,5,6,7,8,9）" << *endl*;

*cout* << "第二组数据：单链表元素为 （1, 1, 2, 2, 2, 3, 4, 5, 5, 5, 6, 6, 7, 7, 8, 8, 9）" << *endl*;

*cout* << "第三组数据：单链表元素为 （1, 2, 3, 4, 5, 5, 6, 7, 8, 8, 9, 9, 9, 9, 9）" << *endl*;

linkedList tst121,tst122,tst123;

tst121.tailList(10);

tst121.delsame(tst121);

tst122.tailList(17);

tst122.delsame(tst122);

tst123.tailList(15);

tst123.delsame(tst123);

*cout*<< "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << *endl*;

/\*测试13\*/

*cout* <<"递增有序单链表L1、L2，不申请新结点，利用原表结点对2表进行合并，" << *endl*;

*cout* <<"并使得合并后成为一个集合，合并后用L1的头结点作为头结点，删除L2的头结点，要求时间性能最好" << *endl*;

*cout* <<"第一组" << *endl*;

*cout* <<"第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）" << *endl*;

*cout* <<"第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）" << *endl*;

*cout* <<"第二组" << *endl*;

*cout* <<"第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）" << *endl*;

*cout* <<"第二个单链表元素为 （2，4，5，7，8，9，12，22）" << *endl*;

*cout* <<"第三组 " << *endl*;

*cout* <<"第一个单链表元素为 （）" << *endl*;

*cout* <<"第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10）" << *endl*;

linkedList tst131, tst132, tst133, tst134, tst135, tst136;

tst131.tailList(10);

tst132.tailList(13);

tst131.AbingB(tst131, tst132);

tst133.tailList(10);

tst134.tailList(8);

tst133.AbingB(tst133, tst134);

tst135.tailList(0);

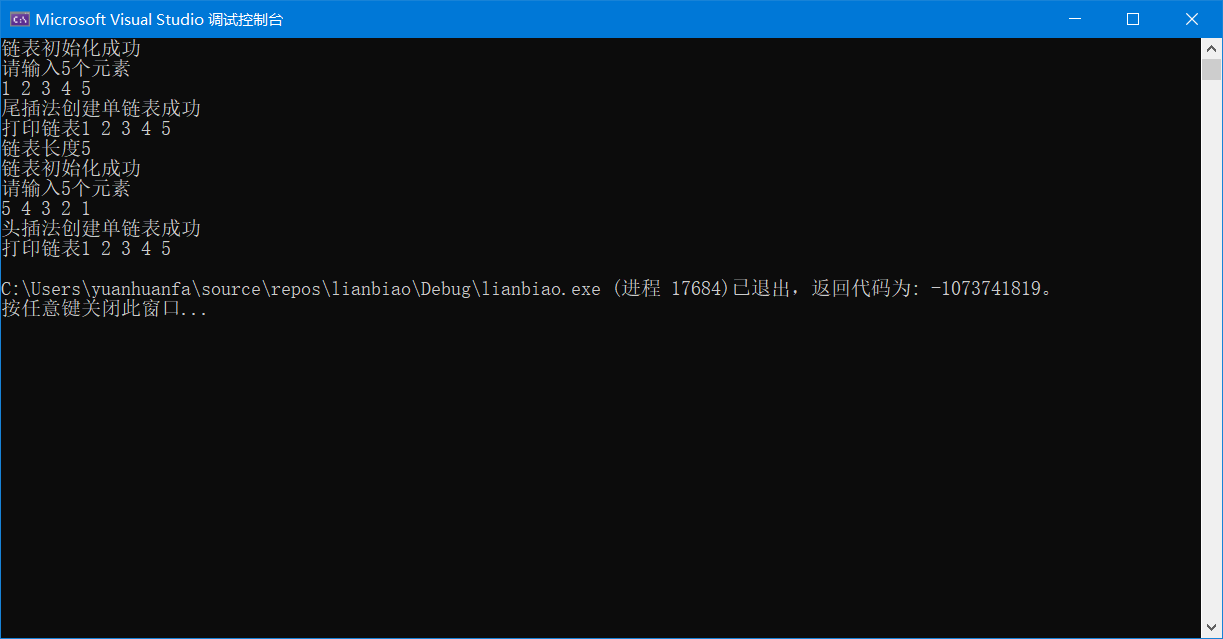
tst136.tailList(10);

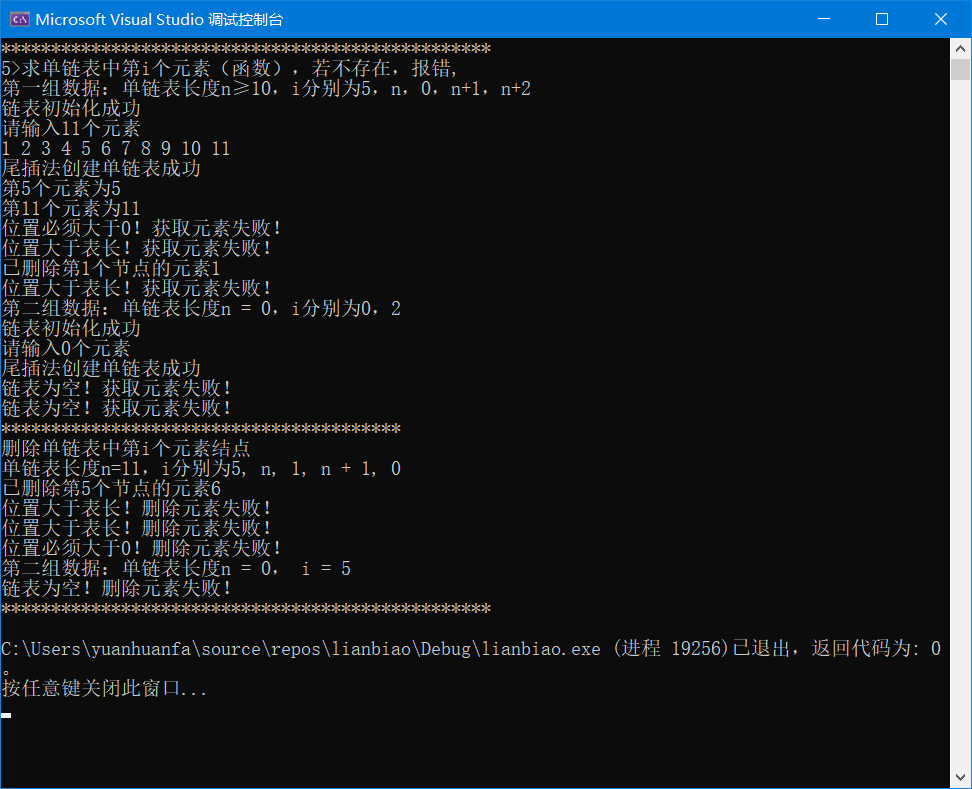
tst135.AbingB(tst135, tst136);

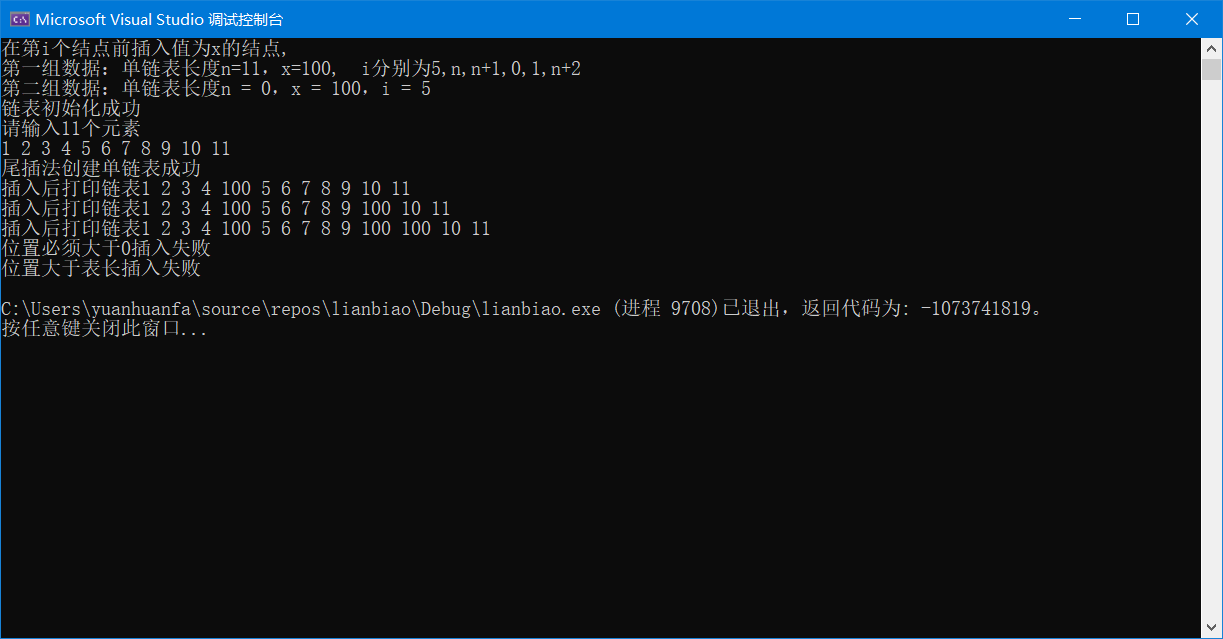
return 0;

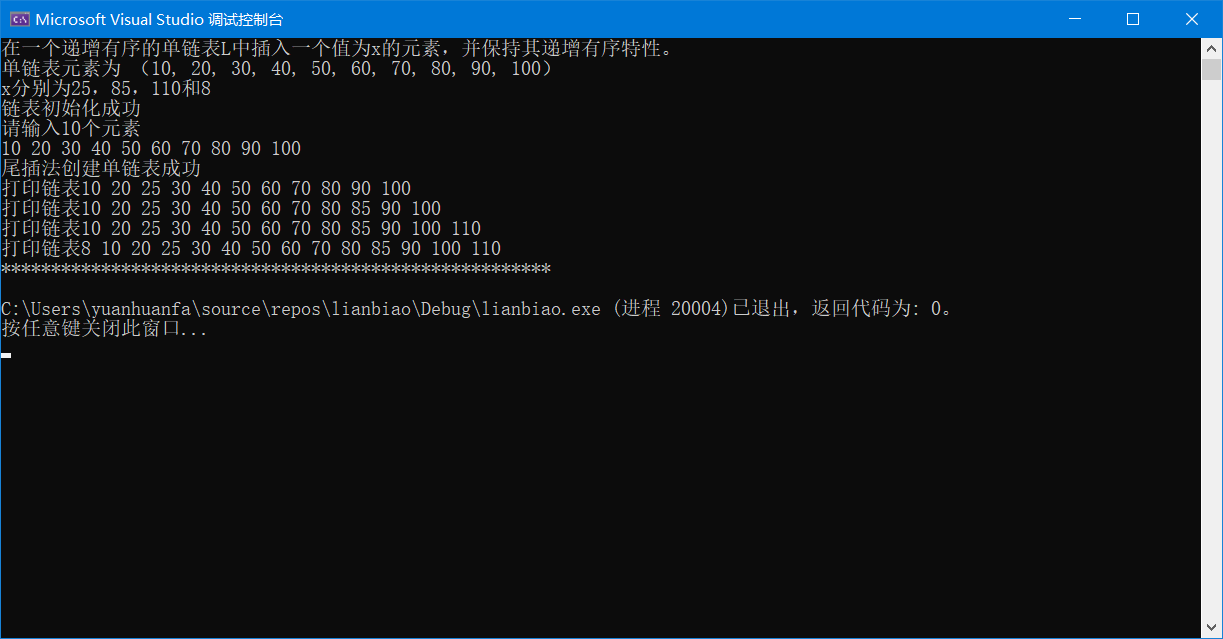
}

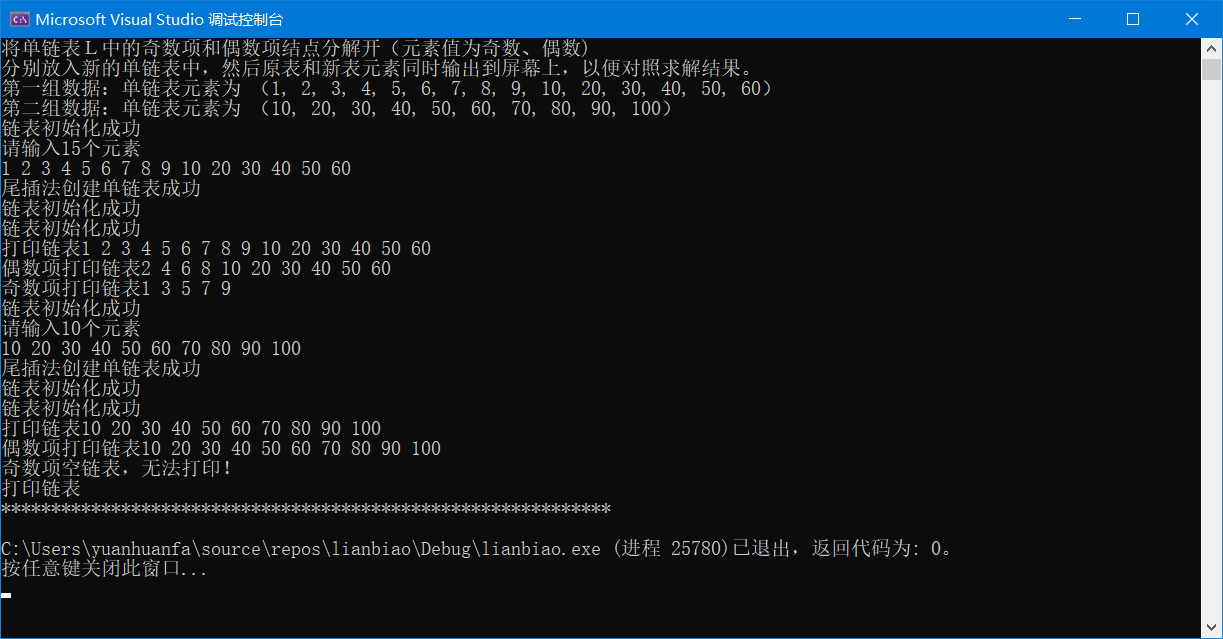
#### （三） 实验截图

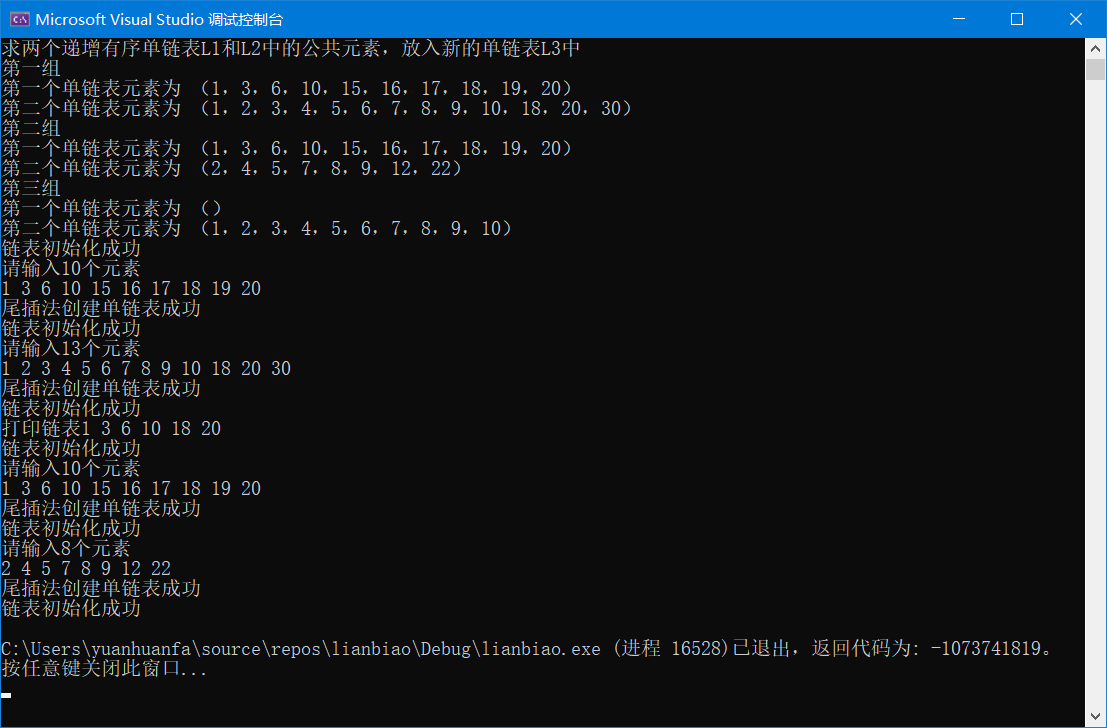


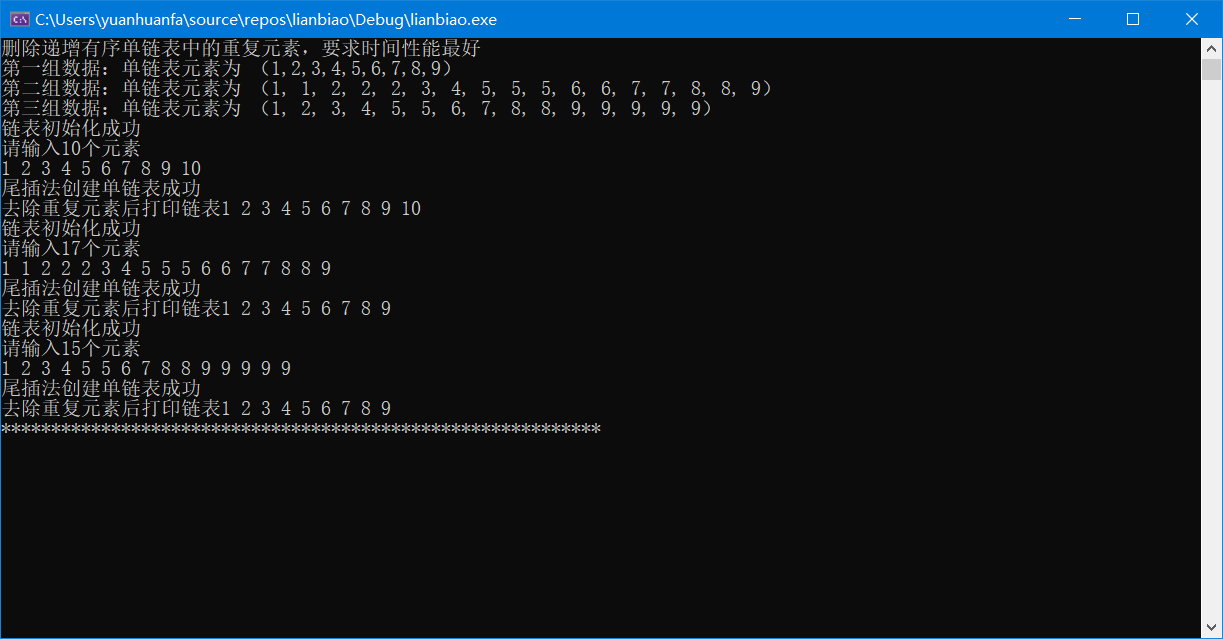


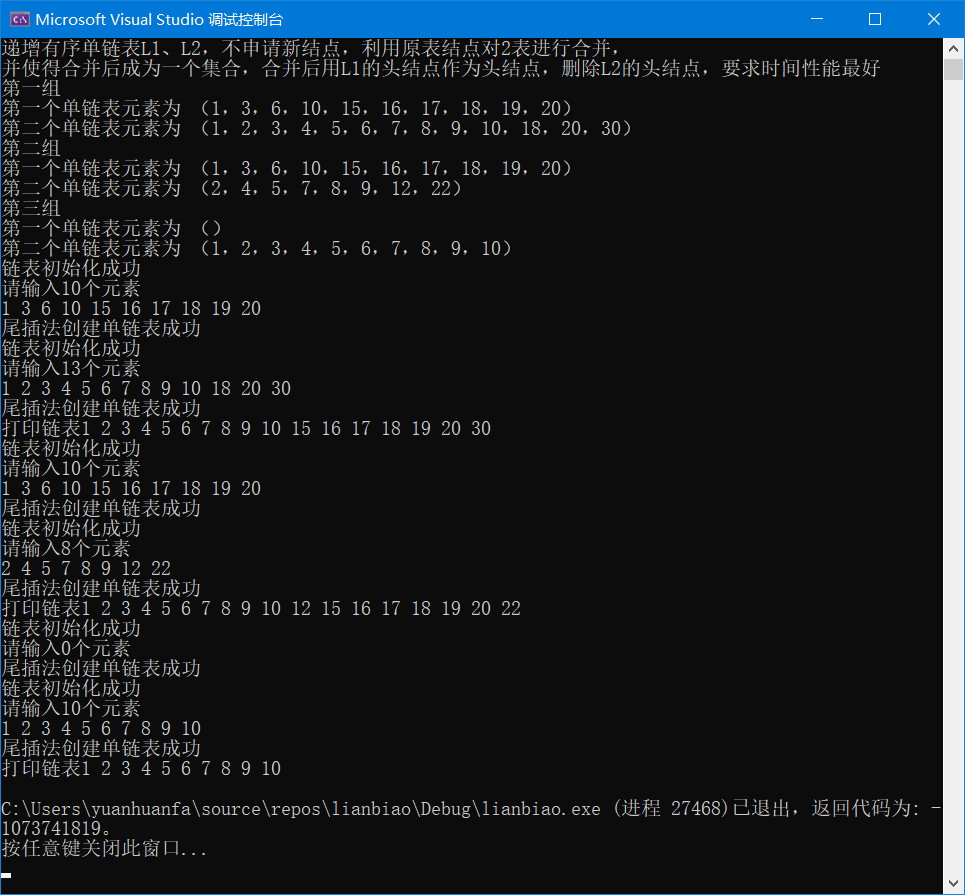












#### （四） 心得体会

这次实验很难，对我来说非常难，要想办法把顺序表时的各种函数化到链表中来应用，

还需要时时刻刻注意节点的位置，避免指错数据的地址，有一个地方不正确就要想办法调试好久，在某个函数中由于忘记给链表初始化导致问题耽搁好久，在将两个单链表公共元素提取出来时，花费了很长时间来将公共元素重新链接起来，在将一个单链表合并到另一个单链表问题上，由于不能创建新的链表，不能创建新的节点，既要考虑去除重复，同时还要考虑如何将元素链接到表1上，我无法从之前的顺序表的操作的函数中得到相关的对策，只能去查阅资料再来写。对于这次实验中我感觉没有解决好的一个是上述的问题，另一个是将两个单链表公共元素放到第三个单链表的问题， 虽然能编译通过，但是总感觉有些地方是存在问题的，只能等对单链表有了进一步的理解才能确定。