**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

一、调试成功程序及说明

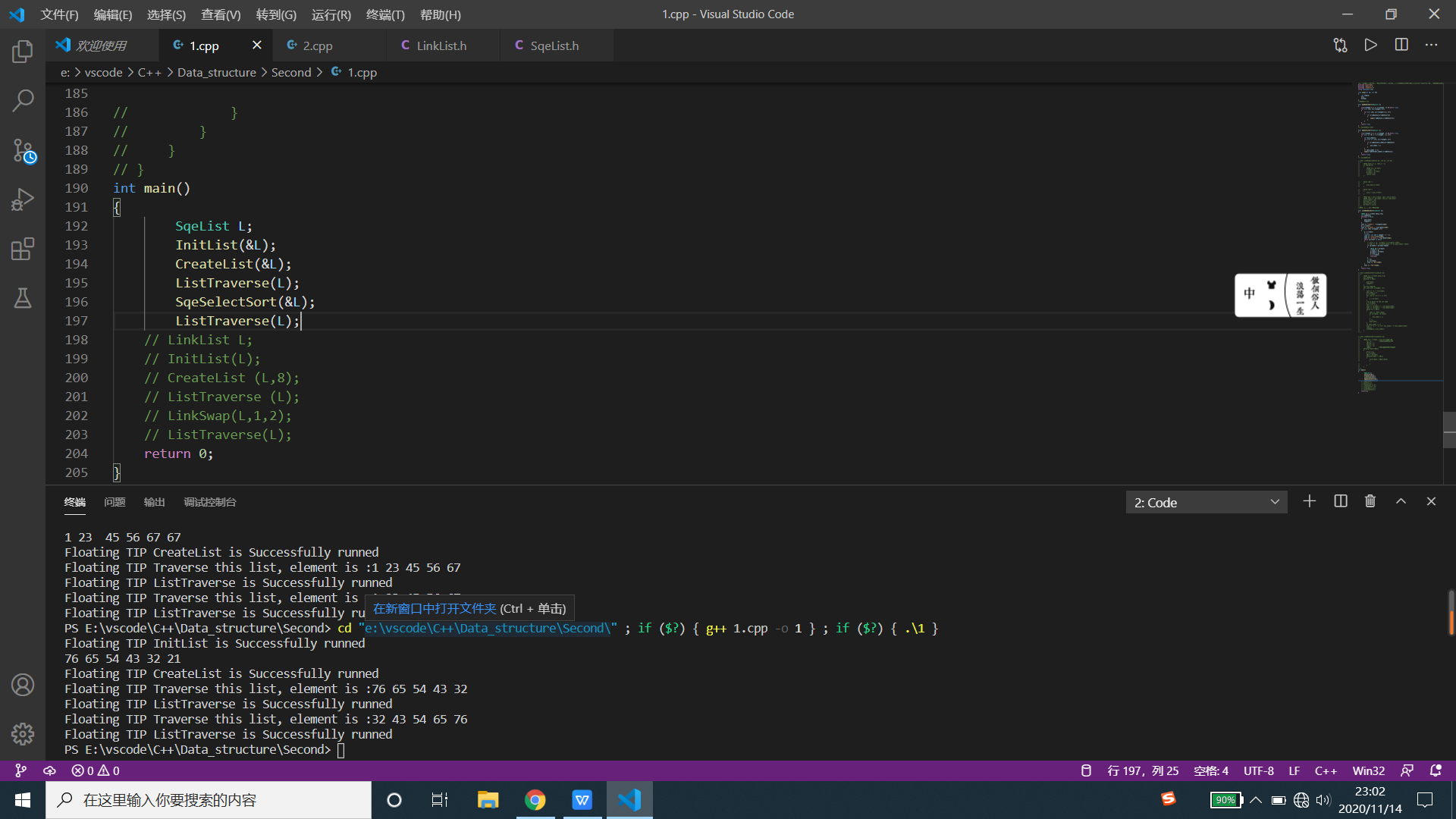
1、

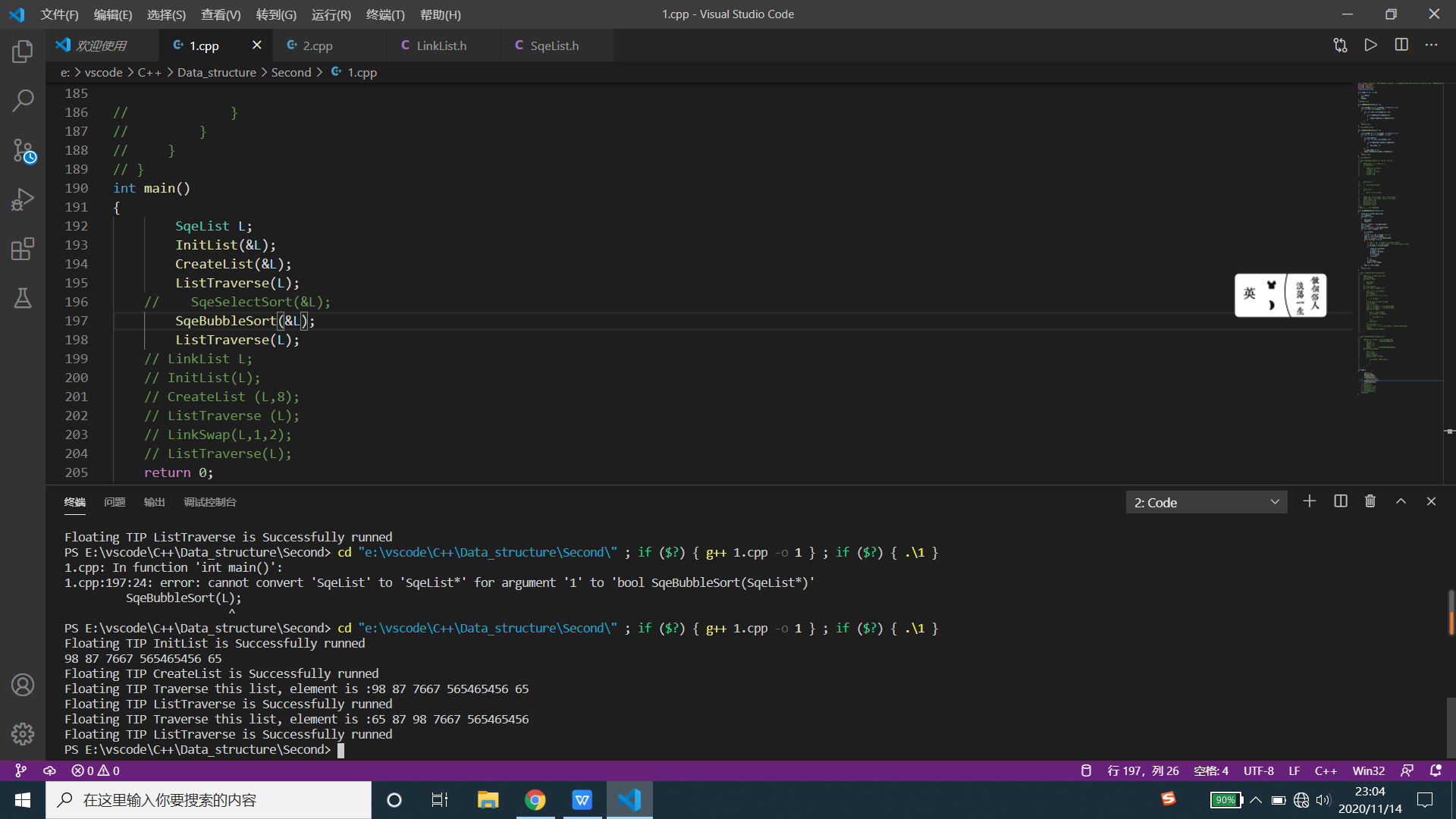
题目：1. 设元素值为整型的线性表L，分别采用顺序结构和链式结构存储，编写函数，用选择/冒泡排序算法实现线性表的表排序。

算法思想：（1）冒泡排序思想:对未排序的数字从一端进行两两比较，每次选出为排序数字的一个最大值，放到当前应排序的一端，循环进行操作。

1. 简单选择排序思想:选出未排序的数据中最值的下标，根据下标进行交换和排序。
2. 对于顺序结构，直接进行数据的交换，对于链式结构，应用指针的实现对结点位置的交换。

运行结果：





结果分析：运行结果正确。

附源程序。

#include <iostream>

#include <stdlib.h>

using namespace std;

typedef struct LNode

{

int data; //数据域

struct LNode \*next; //指针域

}LNode,\*LinkList;

bool InitList(LinkList &L)

{

L = (LNode\*)malloc(sizeof(LNode)); //这里申请了一个头结点,链表指针指向了头结点

if(L == NULL) exit (0);

L->data = 0;

L->next=NULL;

return true;

}

bool DestroyList (LinkList &L)

{

LNode \*p=L->next,\*q=p;

L->next=NULL;

while (p)

{

p=p->next;

free(q);

q=p;

}

free (L);

L = NULL;

return true;

}

//创建一个有n个元素的链式结构的线性表

bool CreateList(LinkList &L,int n)

{

LNode \*p = L;

int a[100]={8,7,6,5,4,3,2,1,4,2,4,3,5,6,7,8,9,2,5,4};

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

{

p->next = (LNode \*)malloc(sizeof(LNode));

p = p->next;

p->data = a[i];

}

p->next = NULL;

return true;

}

bool CreateNaturalList(LinkList &L,int n)

{

LNode \*p = L;

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

{

p->next = (LNode \*)malloc(sizeof(LNode));

p = p->next;

p->data = i;

}

p->next = NULL;

return true;

}

bool CreateListByHand(LinkList &L)

{

int n,t=0;

cout << "Please input the number of this series : ";

cin >> n;

LNode \*p = L;

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

{

p->next = (LNode \*)malloc(sizeof(LNode));

p = p->next;

cin >> t;

p->data = t;

}

p->next = NULL;

return true;

}

bool CreateList\_test1(LinkList &L,int n)

{

LNode \*p = L;

int a[100]={11445,1112,998,965,456,258,154,123,101,99,95,56,55,12,7};

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

{

p->next = (LNode \*)malloc(sizeof(LNode));

p = p->next;

p->data = a[i];

}

p->next = NULL;

return true;

}

bool CreateList\_test2(LinkList &L,int n)

{

LNode \*p = L;

int a[100]={33441,2777,1225,1023,1001,986,952,845,687,514,503,421,333,212,111,28,17,2,-22332};

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

{

p->next = (LNode \*)malloc(sizeof(LNode));

p = p->next;

p->data = a[i];

}

p->next = NULL;

return true;

}

bool CreateList\_test3(LinkList &L,int n)

{

LNode \*p = L;

int a[100]={2,17,25,36,59,98,122,158,199,202,1512,25215,54545,458481,44787454};

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

{

p->next = (LNode \*)malloc(sizeof(LNode));

p = p->next;

p->data = a[i];

}

p->next = NULL;

return true;

}

bool CreateList\_test4(LinkList &L,int n)

{

LNode \*p = L;

int a[100]={-110,-2,36,58,999,1214,2589,6545,6589,6666,8899,10216,11245,56254,5658265};

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

{

p->next = (LNode \*)malloc(sizeof(LNode));

p = p->next;

p->data = a[i];

}

p->next = NULL;

return true;

}

bool ListTraverse (LinkList L)

{

if (L)

{

LNode \*p=L->next;

while (p)

{

cout << p->data << " ";

p = p->next;

}

cout << endl;

return true;

}

else

{

return false;

}

}

//清空链表

bool ClearList (LinkList &L)

{

LNode \*p=L->next,\*q=p;

while (p)

{

p = p->next;

free(q);

q=p;

}

L->next = NULL;

return true;

}

//judge list whether empty

bool ListEmpty (LinkList &L)

{

if ( L->next == NULL) return true; //if list has one node is head

else return false;

}

int ListLength(LinkList &L)

{

if(!L) return -1;

LNode \*p = L->next;

int cnt = 0;

while(p)

{

p = p->next;

cnt ++;

}

return cnt;

}

bool GetElem (LinkList &L, int pos, int &e)

{

int length = ListLength(L);

if (pos<1 || pos > length) return false; //如果数据位置非法，那么就返回操作失败的标志

LNode \*p = L->next;

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

{

p = p->next;

}

e = p->data;

return true;

}

int LocateElem (LinkList &L, int e)

{

LNode \*p = L->next;

int cnt = 1;

while (p)

{

if (p->data == e) return cnt;

p = p->next;

cnt ++ ;

}

return 0;

}

bool PriorElem(LinkList &L, int cur\_e, int &pre\_e)

{

LNode \*p = L->next, \*q = p;

while (p)

{

if (p->data == cur\_e && p != L->next)

{

pre\_e = q->data;

return true;

}

q = p;

p = p->next;

}

return false;

}

bool NextElem(LinkList &L, int cur\_e, int &next\_e)

{

LNode \*p = L->next;

while(p->next)

{

if(p->data == cur\_e)

{

next\_e = p->next->data;

return true;

}

p = p->next;

}

return false;

}

bool SetElem(LinkList &L, int pos ,int &e)

{

int length = ListLength (L);

if (pos<1 || pos > length )

{

cout << "Set position is overflow !"<<endl;

return false; //如果超出范围的话

}

LNode \*p = L->next;

for (int i = 1; i < pos; i++) p = p->next;

int temp = p->data;

p->data = e;

e = temp;

return true;

}

bool InsertElem(LinkList &L, int pos, int e)

{

int length = ListLength (L);

if (pos < 1 || pos > length + 1 ) //可以加在最后一个，所以长度要加一

{

cout << "Set position is overflow !"<<endl;

return false; //如果超出范围的话

}

LNode \*p = L->next,\*q =NULL;

for (int i = 2; i < pos; i ++) p = p->next; //this node is at the position of pos-1

q = (LNode \*)malloc( sizeof(LNode) );

if (pos == 1)

{

q->data = e;

q->next = L->next;

L->next = q;

}

else

{

q->data = e;

q->next = p->next;

p->next = q;

}

return true;

}

bool DeleteElem(LinkList &L, int pos, int &e)

{

int length = ListLength (L);

if (pos < 1 || pos > length ) //可以加在最后一个，所以长度要加一

{

cout << "Set position is overflow !"<<endl;

return false; //如果超出范围的话

}

LNode \*p = L->next,\*q=p;

for (int i = 2; i < pos; i ++) p=p->next; //找到pos前边的一个，就是位置是pos-1

if (pos == 1)

{

e = L->next->data;

q = L->next;

L->next = L->next->next;

free(q);

return true;

}

e = p->next->data; //e是他下一个元素的值

q = p->next;

p->next = q->next;

free(q);

return true;

}

// bool ListReverse(LinkList &L)

// {

// if (L == NULL) return false;

// LNode \*p=L->next;

// LinkList RL = (LNode \*)malloc(sizeof(LNode));

// RL->next = NULL;

// while (p != NULL)

// {

// L->next=p->next;

// p->next=RL->next;

// RL->next=p;

// p = L->next;

// }

// L=RL;

// return true;

// }

#include <iostream>

#include <cstdlib>

using namespace std;

/\*

1．编程实现书P12 ADT List 基本操作14个：

（1）用顺序存储结构实现； （2）用链式存储结构实现；

\*/

const int LISTINITSIZE = 16;

const int LISTICRESIZE = 8;

typedef struct SqeList

{

int \*pData = NULL; //动态储存空间的基地址

int length; //储存元素个数

int size; //当前已分配空间大小

}SqeList;

bool InitList(SqeList \*L) //传入的是引用型，用指针来传

{

L->pData = (int\*)malloc(LISTINITSIZE\*sizeof(int));

if(L->pData == NULL)

{

cout << "Floating TIP Memory overflow"<<endl;

exit(0);

}

L->size = LISTINITSIZE;

L->length = 0;

cout << "Floating TIP InitList is Successfully runned"<<endl;

return true;

}

bool CreateList(SqeList \*L)

{

int n;

//cout << "Please enter the length of this SqeList you want to creat: "<<endl;

//cin >>n;

n=5;

while(n > L->size)

{

int \*temp;

temp = (int \*)realloc (L->pData,LISTINITSIZE\*sizeof(int));

if (temp == NULL) return false;

L->pData = temp;

L->size+=LISTICRESIZE;

cout <<"Floating TIP L.size has increased and now is "<<L->size<<endl;

}

//cout <<"Please enter "<<n<<" elements separated by Spaces :"<<endl;

int test;

int ttt[50]={10,9,8,7,6,5,4,3,2,1,0,-1};

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

{

cin >> test;

L->pData[i] = test;

//L->pData[i]=ttt[i];

L->length+=1;

}

cout << "Floating TIP CreateList is Successfully runned"<<endl;

return true;

}

bool DestroyList(SqeList \*L)

{

if(L->pData != NULL )

{

free(L->pData);

L->pData = NULL;

}

L->length = 0 ;

L->size = 0;

cout << "Floating TIP DestroyList is Successfully runned"<<endl;

return true;

}

bool ClearList (SqeList \*L)

{

L->length = 0;

cout << "Floating TIP ClearList is Successfully runned"<<endl;

return true;

}

//判断是否为一个空表

bool ListEmpty (SqeList L)

{

if (L.length == 0)

{

cout << "Floating TIP ListEmpty is Successfully runned"<<endl;

return true; //表中元素的长度是0,那么就是一个空表

}

cout << "Floating TIP InitList is Successfully runned"<<endl;

return false;

}

int ListLength(SqeList L)

{

cout << "Floating TIP ListLength is Successfully runned"<<endl;

return L.length; //直接返回表的长度

}

bool GetElem (SqeList L,int i,int &e)

{

if (i<1||i>L.length) return 0;

e = L.pData[i-1];

cout << "Floating TIP CreateList is Successfully runned"<<endl;

return true;

}

//从表中找到e这个元素的位置，并返回他的位置信息

int LocateElem (SqeList L, int e)

{

for (int i = 0; i < L.length; i ++ )

{

if(L.pData[i] == e)

{

cout << "Floating TIP LocateElem is Successfully runned"<<endl;

return i+1;

}

}

return -1;

//没找到,返回-1

}

//如果cur\_e在表L中,而且不是第一个元素，那么通过pre\_e返回他前边的一个元素

bool PriorElem (SqeList L, int cur\_e,int &pre\_e)

{

for (int i = 1; i < L.length; i ++)

{

if (L.pData[i]==cur\_e)

{

pre\_e = L.pData[i-1];

return true;

}

}

//如果查找失败，返回False,pre\_e不进行操作

return false;

}

//如果cur\_e在表L中,而且不是最后一个元素，那么通过next\_e返回他后边的一个元素

bool NextElem (SqeList L, int cur\_e,int &next\_e)

{

for (int i = 0; i < L.length - 1; i ++)

{

if (L.pData[i]==cur\_e)

{

next\_e = L.pData[i+1];

return true;

}

}

//如果查找失败，返回False,pre\_e不进行操作

return false;

}

//在屏幕上打印所有的数据,并用空格隔开，最后输出一个空行

bool ListTraverse (SqeList L)

{

if(L.pData == NULL || L.length == 0)

{

return false;

}

cout << "Floating TIP Traverse this list, element is :";

for (int i = 0; i < L.length ; i ++) cout << L.pData[i] << " ";

cout << endl;

cout << "Floating TIP ListTraverse is Successfully runned"<<endl;

return true;

}

//将线性表中第i个元素用e代替，并将旧值用e返回

bool SetElem (SqeList \*L , int i, int &e)

{

if (i < 1 || i>L->length) return false; //i不在可选的范围内

int temp;

temp = L->pData[i-1];

L->pData[i-1] = e;

e = temp;

return true;

}

bool InsertElem (SqeList \*L, int i, int e)

{

if (i < 1 || i > L->length + 1) return false; //i不在可选的范围内

//插入时长度不够,就再申请一点空间

if (L->length + 1 > L->size)

{

int \*temp;

temp = (int \*)realloc (L->pData,LISTINITSIZE\*sizeof(int));

if (temp == NULL) return false;

L->pData = temp;

L->size+=LISTICRESIZE;

}

int temp;

for (int j = L->length ; j > i - 1; j--)

{

L->pData[j]=L->pData[j-1];

}

L->pData[i-1] = e;

L->length +=1;

return true;

}

bool DeleteElem (SqeList \*L, int i, int &e)

{

if (i < 1 || i>L->length) return false; //i不在可选的范围内

e = L->pData[i-1];

for (int j = i - 1; j < L->length-1;j++)

{

L->pData[j] = L->pData[j+1];

}

L->length --;

return true;

}

/\*1. 设元素值为整型的线性表L，分别采用顺序结构和链式结构存储，编写函数，用选择/冒泡排序算法实现线性表的表排序。\*/

#include <iostream>

#include "SqeList.h"

#include "LinkList.h"

using namespace std;

void swap(int &a, int &b)

{

int temp=a;

a=b;

b=temp;

}

//交换辅助程序

bool SqeBubbleSort(SqeList \*L)

{

if(L->length == 1 || L->length == 0)return true;

for (int i=0; i<L->length; i++)

{

for (int j=0; j<L->length-i-1; j++)

{

if (L->pData[j]>L->pData[j+1])

{

swap(L->pData[j],L->pData[j+1]);

}

}

}

return true;

}

//顺序结构的冒泡排序

bool SqeSelectSort(SqeList \*L)

{

if(L->length == 1 || L->length == 0)return true;

for (int i = 0; i < L->length - 1; i++)

{

int min\_index=i;

for (int j = i+1; j<L->length; j++)

{

if (L->pData[min\_index]>L->pData[j])

{

min\_index = j;

}

}

if (min\_index != i)

swap(L->pData[min\_index],L->pData[i]);

}

return true;

}

//顺序结构的选择

// bool LinkSwap(LinkList &L, int dx, int dy)

// {

// LNode \*pre\_x = L, \*pre\_y = L;

// if (dy-dx==1)

// {

// LNode \*m = q->next;

// r->next = m;

// q->next = m->next;

// m->next = q;

// return true;

// }

// while (dx--)

// {

// pre\_x=pre\_x->next;

// }

// while (dy--)

// {

// pre\_y = pre\_y->next;

// }

// LNode \*px = pre\_x->next, \*py = pre\_y->next;

// LNode \*nxt\_x = px->next, \*nxt\_y = py->next;

// pre\_x->next = py;

// py->next = nxt\_x;

// pre\_y->next = px;

// px->next = nxt\_y;

// }

//交换第x,和第y个位置的元素

bool LinkBubbbleSort(LinkList &L)

{

LNode \*p = L->next,\*q=p,\*r=p;

int length=0;

while(p != NULL)

{

p=p->next;

length++;

}

cout << "length = "<<length<<endl;

p = L->next;

cout << "p->data = "<<p->data<<endl;

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

{

q = L->next;

r = L;

register int cnt = length - i - 1;

cout << "cnt="<<cnt<<endl;

cout << "q->data = "<<q->data<<endl;

while (q->next != NULL)

{

// cout << "A : q->data "<< q->data <<endl;

// cout << "B : q->next->data "<< q->next->data <<endl;

if (q->data > q->next->data)

{

LNode \*m = q->next;

r->next = m;

q->next = m->next;

m->next = q;

r = r->next;

continue;

}

r = q;

q = q->next;

cout << "NO"<<endl;

}

cout << "YES"<<endl;

}

return true;

}

// bool LinkSelectSort(LinkList &L)

// {

// LNode \*p = L->next,\*q=p,\*r=p;

// int length=0;

// while(p != NULL)

// {

// p=p->next;

// length++;

// }

// int min\_index=0;

// for (int i=0; i<length; i++)

// {

// cout << "i = "<<i<<endl;

// min\_index = i;

// p = L->next;

// for (int k = 0; k < i; k++)

// {

// p = p->next;

// }

// //p is point to the ist node

// q = p->next;

// int j = i + 1;

// cout << "p->data = " <<p->data<<endl;

// cout << "q->data = " <<q->data<<endl;

// while (q != NULL)

// {

// cout << "YES"<<endl;

// if (p->data > q->data)

// {

// min\_index = j;

// }

// j ++;

// q=q->next;

// }

// if (min\_index != i)

// cout << "i = " << i<<" min\_index = "<<min\_index<<endl;

// cin>>i;

// LinkSwap(L,i,min\_index);

// }

// }

// bool LinkSelectSort(LinkList &L)

// {

// LNode \*p = L->next, //当前应放入的位置

// \*k = p, //k记录要交换的结点

// \*pre\_p = L,

// \*pre\_k = p,

// \*nxt\_k = p,

// \*pMax; //记录最大的数据的指针

// while(p->next!=NULL)

// {

// pre\_k = p;

// k = p->next;

// pMin = p->next;

// while(k->next != NULL)

// {

// if(k->data > pMax->data)

// {

// }

// }

// }

// }

int main()

{

SqeList L;

InitList(&L);

CreateList(&L);

ListTraverse(L);

// SqeSelectSort(&L);

SqeBubbleSort(&L);

ListTraverse(L);

// LinkList L;

// InitList(L);

// CreateList (L,8);

// ListTraverse (L);

// LinkSwap(L,1,2);

// ListTraverse(L);

return 0;

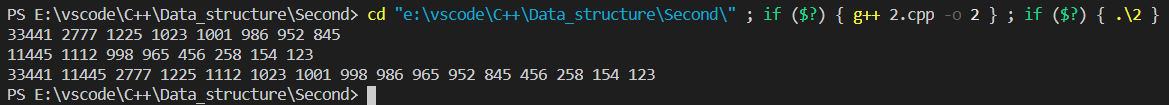
}

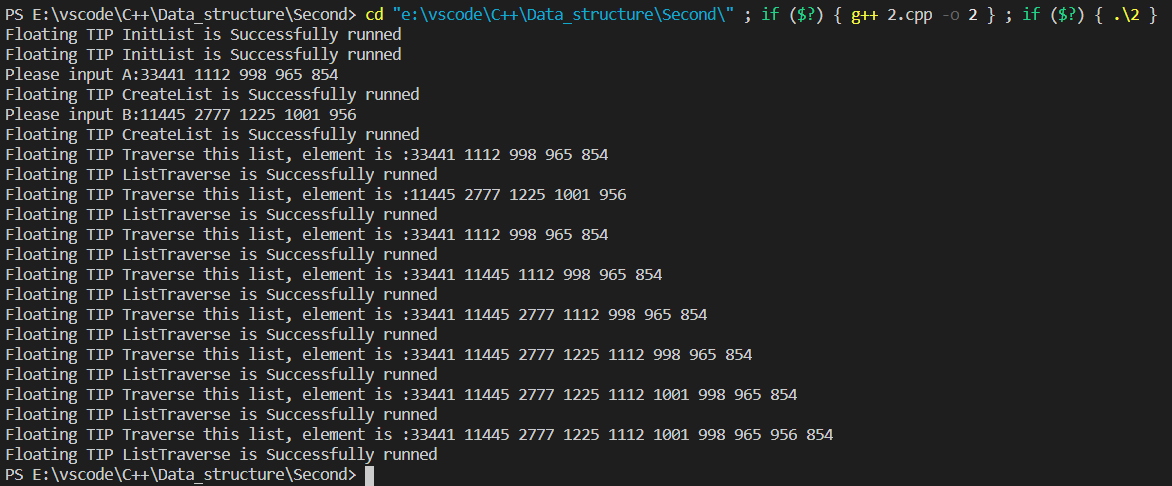
2、

题目：设线性表A、B，元素值为整型，且递减有序，编写函数，实现下列功能：对采用顺序结构和链式结构2种存储结构，要求在A的空间上构成一个新线性表C，其元素为A和B元素的并集，且表C中的元素值递减有序（互不相同）。

算法思想：两个表都为递减有序，要求再A的空间上插入B，那么从B的表头挨个向A中插入元素，共有3种情况，当前B中的元素大于A中的最大值，那么插入在A 的头部，当前B中的元素小于A的最小值，那么插在A 的尾部，（这里特殊考虑，当前元素之后的所有元素也都小于A尾部的元素，可以挨个插入，也可以直接移动），若当前B的元素的大小处于A表的中间，那么在A表的合适位置插入B当前的元素。

运行结果：





结果分析：当输入的序列合法时，线性表合并正确。

附源程序：

/\*设线性表A、B，元素值为整型，且递减有序，编写函数，

实现下列功能：对采用顺序结构和链式结构2种存储结构，

要求在A的空间上构成一个新线性表C，

其元素为A和B元素的并集，且表C中的元素值递减有序（互不相同）\*/

#include "SqeList.h"

#include "LinkList.h"

#include <iostream>

using namespace std;

bool SqeMerge(SqeList \*A, SqeList \*B)

{

int j = 0;

for (int i=0; i < B->length; i++)

{

ListTraverse(\*A);

//cout <<"A->length = "<<A->length<<endl;

//cout << "i = "<<i<<endl;

if (A->pData[0] < B->pData[i])

{

//cout << "Insert on 0"<<endl;

InsertElem(A,1,B->pData[i]);

continue;

}

for (; j < A->length; j++)

{

//cout << "j = "<<j<<endl;

if (j == A->length-1)

{

//cout << "YES"<<endl;

if (A->pData[j]>B->pData[i])

{

InsertElem(A,j+2,B->pData[i]);

break;

}

}

if(A->pData[j]>=B->pData[i]&&A->pData[j+1]<=B->pData[i])

{

InsertElem(A,j+2,B->pData[i]);

//cout << "now length is "<< A->length<<endl;

break;

}

}

}

return true;

}

bool LinkMerge(LinkList &A, LinkList &B)

{

LNode \*p = A->next, \*q = B->next, \*L = B; //L可以看做新的表头

LNode \*r1 = p, \*r2 = q;

while(p != NULL && q != NULL && p->data < q->data)

{

r2 = q->next;

q->next = A->next;

A->next = q;

q = r2;

p = A->next;

}

p = A->next;

// cout <<"A : p->data = "<<p->data<<endl;

// cout <<"B : q->data = "<<q->data<<endl;

while (q != NULL)

{

r2 = q->next;

while(p->data > q->data && p->next->data > q->data && p->next != NULL)

{

p = p->next;

// cout << "After p = p->next"<<endl;

// cout <<"p->data = "<<p->data<<endl;

// cout <<"q->data = "<<q->data<<endl;

if (p->next == NULL)

{

break;

}

}

q->next = p->next;

p->next = q;

q = r2;

}

while(q != NULL &&p->next == NULL)

{

r2 = q->next;

q->next = p->next;

p->next = q;

q = r2;

p = p->next;

}

return true;

}

int main()

{

SqeList A,B;

InitList(&A);

InitList(&B);

cout << "Please input A:";

CreateList(&A);

cout << "Please input B:";

CreateList(&B);

ListTraverse(A);

ListTraverse(B);

SqeMerge(&A,&B);

ListTraverse(A);

// LinkList A,B;

// InitList(A);

// CreateList\_test2(A,8);

// ListTraverse(A);

// InitList(B);

// CreateList\_test1(B,8);

// ListTraverse(B);

// LinkMerge(A,B);

// ListTraverse(A);

return 0;

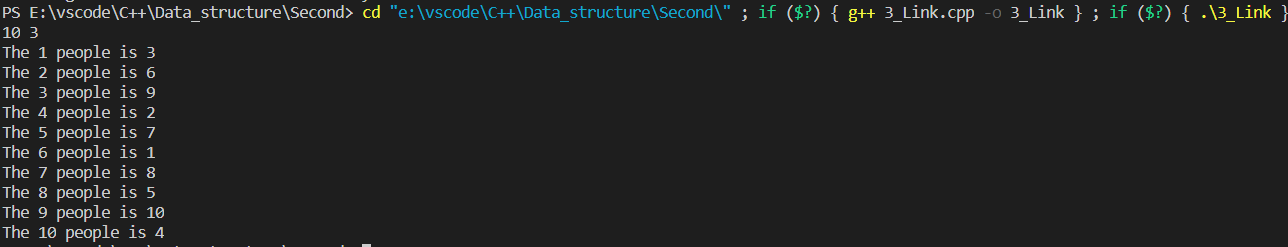
}

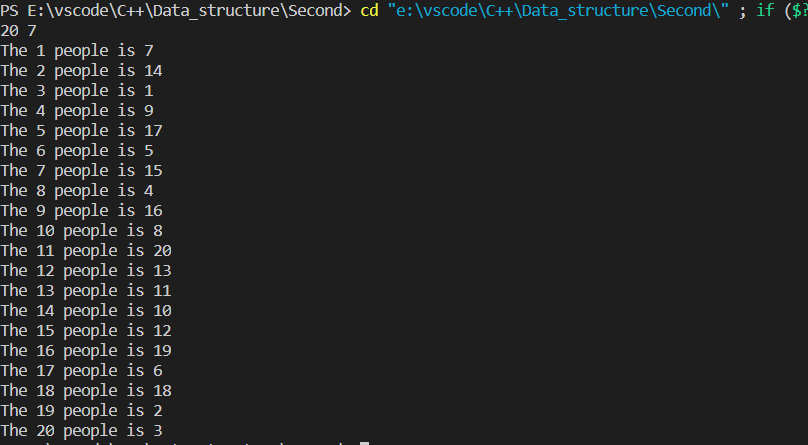
3、

题目：输入正整数n、m（m<n），设有n个人坐成一圈，从第1个人开始循环报数，报到m的人出列，然后再从下一个人开始报数，报到m的人又出列，如此重复，直到所有的人都出列为止。要求用链式结构和顺序结构实现，按出列的先后顺序输出每个人的信息。

算法思想：（1）链式结构。直接挨个进行搜索，每次数到m的时候就删除这个结点，直到节点数为1。注意在开始循环搜索之前将链表转换为循环链表，就是让尾部结点的下一个指向头结点的next。

（2）顺序结构:利用一个bool型的数组记录当前号码对应的人是否还在队列里，false对应在队列里，true对应不在，那么出队的标志就是把对应值置为true。当然，数出m个人的时候，要数在队列里的人，而不是单纯的序号。

运行结果：



结果分析：当输入为（10,3）和（20,7）的时候，运行结果正确。

附源程序。

/\*输入正整数n、m（m<n），设有n个人坐成一圈，从第1个人开始循环报数，报到m的人出列，

然后再从下一个人开始报数，报到m的人又出列，如此重复，直到所有的人都出列为止。

要求用链式结构和顺序结构实现，按出列的先后顺序输出每个人的信息。\*/

#include <iostream>

#include "LinkList.h"

using namespace std;

void print(LinkList L)

{

LNode \*p = L->next,\*q = p;

do

{

cout << p->data << " ";

p = p->next;

}while (p != q);

}

void work(LinkList &L, int n, int m)

{

LNode \*p=L->next, \*q = p;

while (p->next != NULL)

{

p = p->next;

}

p->next = L->next;//制作循环链表

p = L->next; //p指向头结点的下一个

int num=n;

while (num > 1)

{

register int cnt= m-2;

while (cnt--)

{

p = p->next;

}

q = p->next;

num--;

cout << "The "<<n-num<<" people is "<<q->data<<endl;

p->next = q->next;

free(q);

p = p->next;

}

cout << "The "<<n<<" people is "<<p->data<<endl;

return;

}

int main()

{

int n=0, m=0;

cin>>n>>m;

LinkList L;

InitList(L);

CreateNaturalList(L,n);

work(L,n,m);

return 0;

}

/\*输入正整数n、m（m<n），设有n个人坐成一圈，从第1个人开始循环报数，报到m的人出列，

然后再从下一个人开始报数，报到m的人又出列，如此重复，直到所有的人都出列为止。

要求用链式结构和顺序结构实现，按出列的先后顺序输出每个人的信息。\*/

#include <iostream>

#include <string.h>

#include "SqeList.h"

using namespace std;

bool map[10005];

int main()

{

int n = 0, m = 0;

cin >> n >> m;

int num = 0, p = 1; //p相当于一个指针

while (num < n)

{

register int cnt = 0;

while(cnt < (m-1))

{

if (map[p]) //要是被标记过了，直接跳过

{

p++; //下移一个

if(p > n) //要是越界了就循环一个

{

p = 1;

}

}

else

{

cnt++; //要是没标记过就记数

p++;

if (p > n)

{

p = 1;

}

}

}

//找到了这个数

while (map[p])

{

p++;

if (p > n)

{

p = 1;

}

}

num++;

cout << "The "<<num<<" people is "<< p << endl;

map[p] = true;

p++;

if (p > n)

{

p = 1;

}

}

return 0;

}

4、

题目：题目描述：小明在他的果园里种了一些苹果树，这些苹果树排列成一个圆。为了保证苹果的品质，在种植过程中要进行疏果操作。为了更及时地完成疏果操作，小明会不时地检查每棵树的状态，根据需要进行疏果。检查时，如果发现可能有苹果从树上掉落，小明会重新统计树上的苹果个数（然后根据之前的记录就可以判断是否有苹果掉落了)。在全部操作结束后，请帮助小明统计相关的信息。

输出只有一行，包含三个整数T、D、E。其中，

1. T为全部疏果操作结束后所有苹果树上剩下的苹果总数（假设每棵苹果树在最后一次统计苹果个数操作后苹果不会因为疏果以外的原因减少);
2. D为发生苹果掉落的苹果树的棵数;
3. E为相邻连续三棵树发生苹果掉落情况的组数。

算法思想：（1）剩余苹果数只要找到最后一次输入为正的数，减去后边的负数即可（最后数果子，再减去疏果数）

1. 发生苹果掉落的个数。先循环每一棵树，搜索两个正数之间的差值是不是等于中间负数和的绝对值。
2. 将掉落的苹果树的标号记录下来，只要找到三棵相邻的即可，注意循环搜索，就是n后边的是1。

运行结果：



结果分析：运行结果正确,csp模拟测试通过

附源程序：#include <iostream>

#include <fstream>

using namespace std;

int N=0;

int T=0,E=0,D=0;

int t[1005];

bool e[1005];

int m[1005][1005];

int main()

{

ifstream file("4.txt",ios::in);

if(!file)

{

cout << "File can't be opened!"<<endl;

exit(0);

}

file>>N;

//cin>>N;

int mi=0,temp=0;

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

{

//cin>>mi;

file>>mi;

for(int j=1; j<=mi; j++)

{

file>>m[i][j];

//cin>>m[i][j];

}

for(int j=mi; j>=1; j--)

{

if (m[i][j]<=0)

{

t[i]+=m[i][j];

}

else

{

t[i]+=m[i][j];

T+=t[i];

break;

}

}

int now = 0, sub = 0;

for(int j=1; j<=mi; j++)

{

if(m[i][j]>0)

{

if(now == 0)

{

now = m[i][j];

continue;

}

else

{

if (m[i][j]!=now+sub)

{

E++;

e[i]=1;

// cout <<"m[i][j] = " << m[i][j]<<endl;

// cout << "i = "<<i<<endl;

break;

}

now = m[i][j];

sub = 0;

}

}

else if(m[i][j]<=0)

{

sub+=m[i][j];

}

}

}

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

{

bool a1,a2,a3;

a1 = e[i];

if (i < N-1)

{

a2 = e[i+1];

a3 = e[i+2];

}

else if (i == N-1)

{

a2 = e[i+1];

a3 = e[1];

//cout << i<<" "<<a2 << a3 <<endl;

}

else

{

a2 = e[1];

a3 = e[2];

}

if (a1&&a2&&a3)

{

D++;

}

// cout << "i = "<<i<<" ";

// cout << "e[i] = "<<e[i]<<" ";

// cout << "e[((i+1)%(N+1))+1] = "<<e[(i+1)%(N+1)+1]<<" ";

// cout << "e[((i+2)%(N+1))+1] = "<<e[(i+2)%(N+1)+1]<<endl;

// else

// {

// cout << "i = "<<i<<" ";

// cout << a1 <<" "<< a2 <<" "<<a3<<endl;

// }

}

//cout << "e[5] = " <<e[4]<<endl;

cout <<T<<" "<<E<<" "<<D<<endl;

return 0;

}

// N = 4

// 1 2 3 4

// e = 3

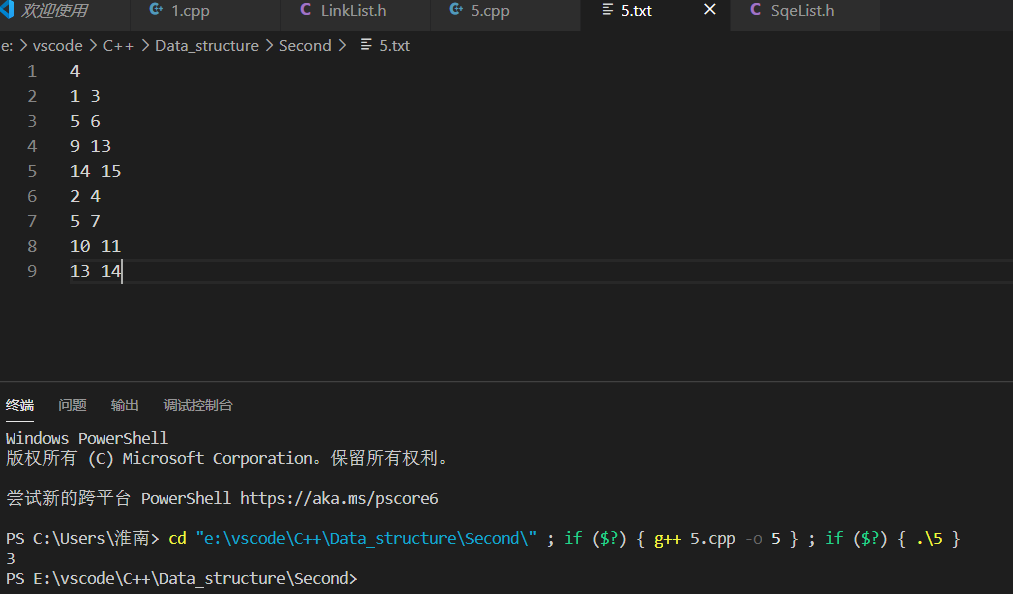
// e + 1 = 4

// e + 2 = 5

5、

题目：问题描述：小H和小W来到了一条街上，两人分开买菜，他们买菜的过程可以描述为，去店里买一些菜然后去旁边的一个广场把菜装上车，两人都要买n种菜，所以也都要装n次车。具体的，对于小H来说有n个不相交的时间段[a1,b1],[a2,b2]…[an,bn]在装车，对于小W来说有n个不相交的时间段[c1,d1],[c2,d2]…[cn,dn]在装车。其中，一个时间段[s, t]表示的是从时刻s到时刻t这段时间，时长为t-s。  
算法思想：注意结果是要求计算这个时间段，开一个数组，将第一个人的时间离散化的表示出来，开始时刻和中间的时间用1来表示，结尾用2来表示。再输入第二个人的时间，搜索结果是答案时间的标志是上一个人在这个点的标志是1，而且相邻下一个点的标志是1或者2（连续或者达到结束的时间，1和2分开这样是为了误把结束时间当做开头开始算起）

运行结果：



结果分析：

运行结果正确，通过csp模拟考试的测试。

附源程序:

#include <iostream>

#include <cstdlib>

#include <fstream>

using namespace std;

int n,ans=0;

int a[1000005];

int main()

{

fstream file ("5.txt",ios::in);

int h,t;

file >> n;

//cin >> n;

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

{

file >> h >>t;

//cin >> h >> t;

for (int j = h + 1; j < t; j++)

{

a[j] = 1;

}

a[h] = 1;

a[t] = 2;

}

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

{

file >> h >> t;

//cin >> h >> t;

for (int j=h; j<t; j++)

{

if (a[j] == 1&&(a[j+1] == 1 || a[j+1] == 2))

{

ans++;

}

}

}

cout << ans;

return 0;

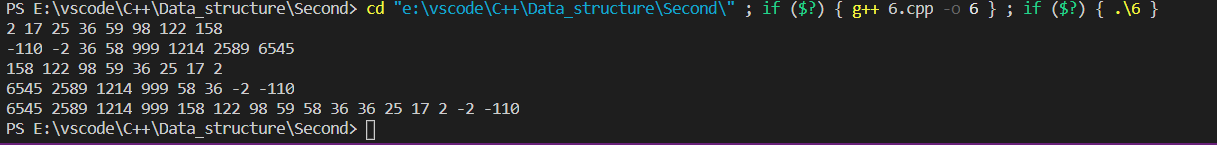
}

6、

题目：两个递增的线性表A和B，合并成递减的线性表C

算法思想：先设计一个逆序的函数，将函数进行逆序，调用第二题的降序合并，则得到结果。

运行结果：



结果分析：运行结果正确，实现了函数要求的功能呢

附源程序。#include <iostream>

#include "LinkList.h"

using namespace std;

bool LinkMerge(LinkList &A, LinkList &B)

{

LNode \*p = A->next, \*q = B->next, \*L = B; //L可以看做新的表头

LNode \*r1 = p, \*r2 = q;

while(p != NULL && q != NULL && p->data < q->data)

{

r2 = q->next;

q->next = A->next;

A->next = q;

q = r2;

p = A->next;

}

p = A->next;

// cout <<"A : p->data = "<<p->data<<endl;

// cout <<"B : q->data = "<<q->data<<endl;

while (q != NULL)

{

r2 = q->next;

while(p->data > q->data && p->next->data > q->data && p->next != NULL)

{

p = p->next;

// cout << "After p = p->next"<<endl;

// cout <<"p->data = "<<p->data<<endl;

// cout <<"q->data = "<<q->data<<endl;

if (p->next == NULL)

{

break;

}

}

q->next = p->next;

p->next = q;

q = r2;

}

while(q != NULL &&p->next == NULL)

{

r2 = q->next;

q->next = p->next;

p->next = q;

q = r2;

p = p->next;

}

return true;

}

//两个递减序列进行合并

bool ListReverse(LinkList &L)

{

if (L == NULL) return false;

LNode \*p=L->next;

LinkList RL = (LNode\*)malloc(sizeof(LNode));

RL->next = NULL;

while (p != NULL)

{

L->next=p->next;

p->next=RL->next;

RL->next=p;

p = L->next;

}

L->next = RL->next;

return true;

}

//逆序的函数

int main()

{

LinkList A,B;

InitList(A);

InitList(B);

CreateList\_test3(A,8);

CreateList\_test4(B,8);

ListTraverse(A);

ListTraverse(B);

ListReverse(A);

ListReverse(B);

ListTraverse(A);

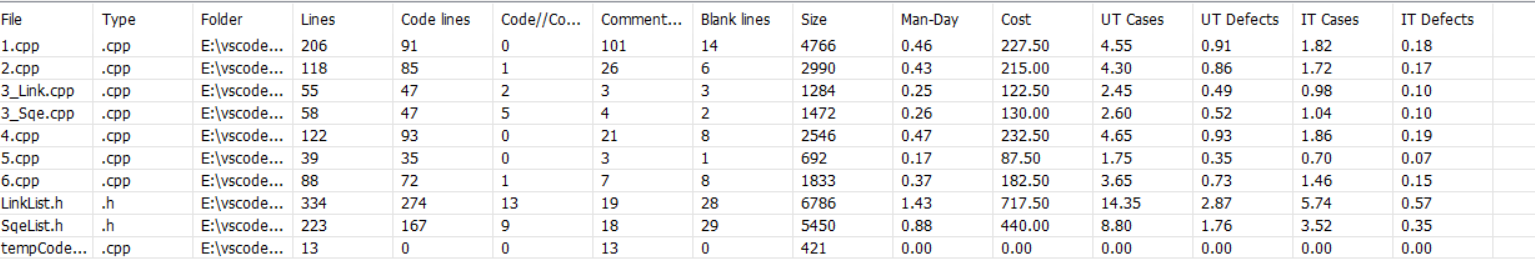
ListTraverse(B);

LinkMerge(A,B);

ListTraverse(A);

}

三、代码行数小结



总行数:1256