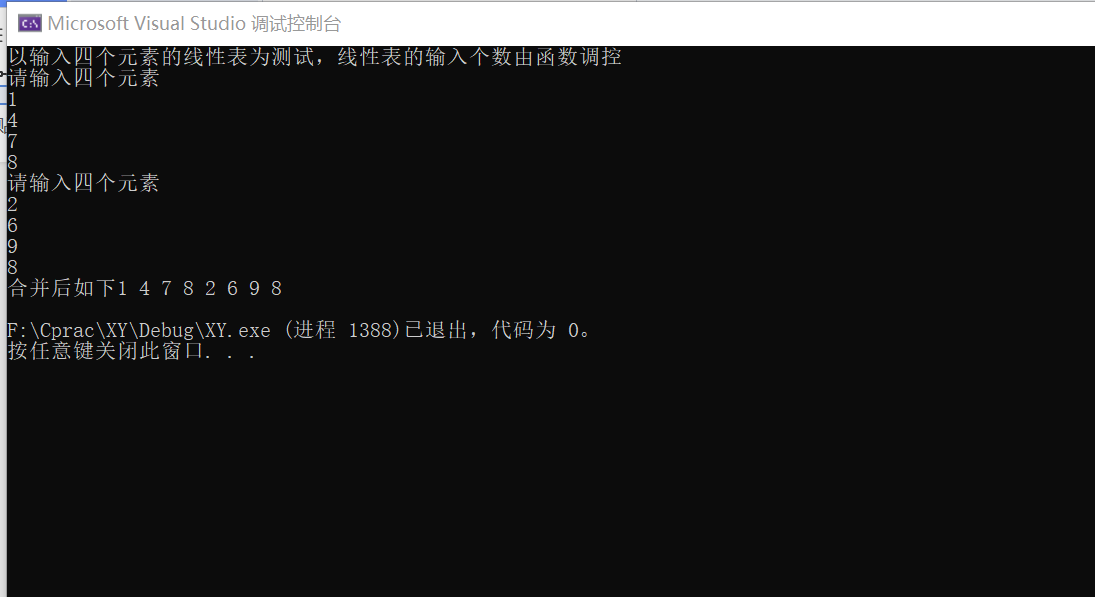
合并顺序表



#include< stdlib.h>

#include <iostream>

using namespace std;

typedef int Status;

typedef int ElemType;

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREAMENT 10

typedef struct { //定义线性表的基本结构

ElemType\* elem; //顺序表第x个元素

int length;

int listsize;

}Sqlist;

Status InitList(Sqlist& L) {

L.elem = (ElemType\*)malloc(LIST\_INIT\_SIZE \* sizeof(ElemType));

if (!L.elem) exit(OVERFLOW);

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

return OK;

}//初始化线性表

Status GetElem(Sqlist& L, int i, ElemType& e) { //将读到的元素返回给 e

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

e = \*(L.elem + i - 1);

return OK;

}//读表元

Status ListInsert(Sqlist& L, int i, ElemType e) {

ElemType\* q, \* p;

ElemType\* newbase = NULL;

if (i<1 || i>L.length + 1) // i值不合法

return ERROR;

if (L.length >= L.listsize)

{

// 当前存储空间已满,增加分配，并插入元素

newbase = (ElemType\*)realloc(L.elem, (L.listsize + LISTINCREAMENT) \* sizeof(ElemType));

if (!newbase)

exit(OVERFLOW); // 存储分配失败

L.listsize += 10; // 增加存储容量

q = &(L.elem[i - 1]);

for (p = &(L.elem[L.length - 1]); p >= q; --p)

\*(p + 1) = \*p;

\*q = e; //插入元素

++L.length;

}

//此为存储空间足够的情况

q = &(L.elem[i - 1]);

for (p = &(L.elem[L.length - 1]); p >= q; --p)

\*(p + 1) = \*p;

\*q = e;

++L.length;

return OK;

}

Status Input\_List(Sqlist& L, int n) {

if (n > L.listsize) return OVERFLOW;

else

{

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

{

cin>>L.elem[i];

L.length++;

}

return OK;

}

}

Status Output(Sqlist L) {

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

cout<<L.elem[i]<<" ";

}

cout << endl;

return OK;

}

void Merge(Sqlist La, Sqlist Lb, Sqlist& Lc) { //将两个表合并成新的表

int i, j;

i = 0;

j = 0;

ElemType\* pc;

Lc.listsize = Lc.length = La.length + Lb.length;

pc = Lc.elem = (ElemType\*)malloc(Lc.listsize \* sizeof(ElemType)); //分配地址

if (!Lc.elem) exit(OVERFLOW);

while (i < La.length) { //将La放到Lc

Lc.elem[i] = La.elem[i];

i++;

}

while (j<Lb.length) //将Lb放到Lc

{

Lc.elem[i] = Lb.elem[j];

i++;

j++;

}

}

int main(void) {

cout<<"以输入四个元素的线性表为测试，线性表的输入个数由函数调控\n";

Sqlist ts;

InitList(ts);

cout << "请输入四个元素\n";

Input\_List(ts, 4);

//Output(ts);

Sqlist tpps;

InitList(tpps);

cout << "请输入四个元素\n";

Input\_List(tpps, 4);

Sqlist dc;

Merge(ts, tpps, dc);

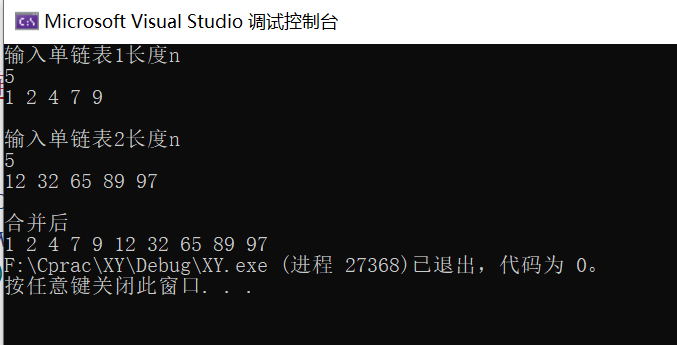
printf("合并后如下");

Output(dc);

return 0;

}

合并单链表



#include<iostream>

using namespace std;

typedef struct node {

int data;

node\* next;

}List, \* Xlist;

void InitXlist(Xlist& l) {

l = new List;

l->next = NULL;

}

void CreatXlist(Xlist& l, int x) {

int k;

Xlist node, rear;

rear = l;

for (int i = 0; i < x; i++) {

cin >> k;

node = new List;

node->data = k;

//rear = l;

rear->next = node;

rear = node;

rear->next = NULL;

}

}

void output(Xlist l) {

while (l->next != NULL) {

cout << l->next->data << " ";

l = l->next;

}

}

int Insert(Xlist l, int i, int val) {

if (!l)

return -1;

Xlist t = new List;

for (int d = 0; d < i - 1; d++) {

l = l->next;

}

t->data = val;

t->next = l->next;

l->next = t;

return 1;

}

void merge(Xlist la, Xlist lb, Xlist &lc) {

Xlist pc = lc;

pc->next = new List;

pc = pc->next;

//cout << &(pc->next) << endl;

while (la->next != NULL) { //a插入

pc->data = la->next->data;

//cout << pc->data << " ";

la = la->next;

pc->next = new List;

pc = pc->next;

}

while (lb->next != NULL) { //将lb加入

pc->data = lb->next->data;

//cout << pc->data << " ";

lb = lb->next;

if (lb->next == NULL) { //处理最后的结点

pc->next = NULL;

break;

}

pc->next = new List;

pc = pc->next;

}

}

int main() {

cout << "输入单链表1长度n" << endl;

int x;

cin >> x;

Xlist test;

InitXlist(test);

CreatXlist(test, x);

// output(test); 1 2 4 7 9 12 32 65 89 97

cout << endl;

cout << "输入单链表2长度n" << endl;

cin >> x;

Xlist testTwo;

InitXlist(testTwo);

CreatXlist(testTwo, x);

//output(testTwo);

cout << endl;

Xlist testThree;

InitXlist(testThree);

merge(test, testTwo, testThree);

cout << "合并后" << endl;

//cout << testThree->next->data;

output(testThree);

return 0;

}