```
#include "stdafx.h"

//线性表的基本操作
/*
   //初始化线性表
   InitList(&L); ------>已在com.cpp文件里定义过
*/


//销毁线性表
int DestroyList(sqList *&L) {
        free(L);
        printf_s("sqList are destroied!!! \tlength==%d\n", L->length);
        return 1;
}


//将线性表重置为空
int ClearList(sqList &L) {
        L.length = 0;
        return 1;
}


//判断线性表示不是空的
bool ListIsEmpty(sqList L) {
        return L.length == 0;
}


//返回L中数据元素的个数
```

```
int ListLength(sqList L) {
        return sizeof(L.data) / L.data[0];
}


//用value返回线性表L中第index个数据元素的值,
bool GetElem(sqList L, int index, int &value) {
        if (index < 1 || index > L.length) {
                return false;
        }
        value = L.data[index - 1];
        return true;
}


//删除L中第index个数据元素, 并用e返回其值
bool ListDelete(sqList &L, int index, int &e) {
        if (index < 1 || index > L.length) {
                return false;
        }
        e = L.data[index - 1];
        for (int i = index - 1; i < L.length; i++) {
                L.data[i] = L.data[L.length - i - 1];
        }
        return true;
}
```

```
#include "stdafx.h"

/*******************王道2019年chapter-2.2 ，应用题第1题（P18）
*********************************
1、题目:
        从顺序表中删除具有最小值的元素（假设唯一），并由函数返回被删除的元素的值，空出的位置由最后一个元素填补，
        若顺序表为空，则显示出错误信息并退出;

2、算法思想:
1°：若传入的顺序表为空则返回错误表示;
2°：通过全盘扫面在这个线性表中找出最小值，并记录他的下标为minIndex;
3°：将线性表最后一个元素放到下表为minIndex的位置上，并把线性表长度-1;
*****************************************************************************************
*/

/*
@param L 顺序表L
@param min 找到的最小值
*/
int DeleteMin(sqList &L, int &min) {
        if (L.length == 0) {
                return -1;
        }
        min = L.data[0];
        int minIndex = 0;
        for (int i = 0; i < L.length; i++) {
                if (L.data[i] < min) {
                        min = L.data[i];
                        minIndex = i;
                }
```

```
        }
        L.data[minIndex] = L.data[L.length - 1];
        L.length--;
        return 1;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.2 ，应用题第2题 (P18)
*********************************
1、题目:
        设计一个高效的算法，将顺序表中所有元素逆置，要求算法的空间复杂度为0(1);

2、算法思想:
1°：若传入的顺序表为空则将引用变量result=-1，表示返回逆置失败，直接退出;
2°：定义i，j两个变量，i从下标0开始，j从下表为L.length-1开始，从两边往中间遍历，并交换他们位置上的元素，
        直到i>=j时，退出循环;
3°：结束后，用传入的引用变量result返回逆置成功的提示;
*****************************************************************************************
*/


void InverseList(sqList &L, int &result) {
        if (L.length == 0) {
                result = -1;
                return;
        }
        int i, j, temp;
        for (i = 0, j = L.length - 1; i < j; i++, j--) {
                temp = L.data[i];
                L.data[i] = L.data[j];
                L.data[j] = temp;
        }
        result = 1;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.2 ，应用题第3题 (P18)
*********************************
```

1、题目:

长度为n的顺序表L，编写一个时间复杂度为O(n)，空间复杂度为O(1)的算法，删除线性表中所有值为x的数据元素;

2、算法思想:

1°：若传入的顺序表为空则返回-1，表示失败标志;

2°：定义一个K变量，用来记录所有值不等于x的元素的个数;全盘扫描式遍历，若dada[i]不等于x，则将k++;

若等于，则k的值保持不变，这样，直到下次遇到一个值为不为x的元素时，直接覆盖掉K的位置即可;

3°：结束后，返回1表示成功的标志;

```
*********************************************************************************************
*/

int DeleteX(sqList &L, int x) {
	if (L.length == 0) {
		return -1;
	}
	int k = 0;
	for (int i = 0; i < L.length; i++) {
		if (L.data[i] != x) {
			L.data[k] = L.data[i];
			k++;
		}
	}
	L.length = k;
	return 1;
```

}

```
#include "stdafx.h"

/*******************王道2019年chapter-2.2 ，应用题第4题（P18）
*********************************
1、题目:
        从有序表中删除其值在给定s与t之间的（要求是s<=x<t）的所有元素有，如果s或t不合理或者顺序表为空，则显示出错误信息并退出;

2、算法思想:
1°：若传入的s与t不合理或表为空，则返回-1，表示失败标志;
2°：定义i，j两个变量，i=s-1，j=t-1，即s至t是指：包括s，但不包括t之间的数据元素；然后依次用t后面的元素
        来覆盖这些被替换位置的元素，左后将线性表的长度置为L.length-(t-s) 即可。
3°：结束后，返回1表示成功的标志;
*****************************************************************************************
*/


int Delete_s_t(sqList &L, int s, int t) {
        if (t > L.length+1 || s > t || L.length == 0) {
                return -1;
        }
        for (int i = s - 1, j = t-1; j < L.length; i++, j++) {
                L.data[i] = L.data[j];
                //printf("i==%d,  j==%d\n",i,j);
        }
        L.length = L.length - (t - s);
        return 1;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.2 ，应用题第4题（P18）
*********************************
1、题目：
        从顺序表中删除其值在给定s与t之间的（要求是s<=x<t）的所有元素有，如果s或t不合理或者顺序表为空，则显示出错误信息并退出；

2、算法思想：
1°：若传入的s与t不合理或表为空，则返回-1，表示失败标志；
2°：定义一个变量k,用来记录元素值在s¯t之间的个数，从前向后扫描顺序表L，若当前元素值不在s¯t，则将元素向前移动k个位置；
        若在，则让k++,
3°：结束后，返回1表示成功的标志；
*****************************************************************************************
*/

//ooo--> out of order
int Delete_s_t_ooo(sqList &L, int s, int t) {
        if (t > L.length + 1 || s > t || L.length == 0) {
                return -1;
        }
        int k = 0;
        for (int i = 0; i < L.length; i++){
                if (L.data[i] >=s && L.data[i] < t) {
                        k++;
                }else {
                        L.data[i-k] = L.data[i];
                }
        }
        L.length -= k;
```

```
        return 1;
}
```

```
#include "stdafx.h"


/*********************王道2019年chapter-2.2 ，应用题第6题（P18）
*********************************
1、题目:
        从有序顺序表中删除所有其值重复的元素，使表中所有元素的值各不相同;

2、算法思想:
1°：若传入的线性表为空，则返回-1，表示失败标志;
2°：因为是有序表，所以他们值相同的元素一定是连续的，将初始时第一个元素看作非重复的有序表，之后以此判断
        后面的元素是否与前面的非重复有序表的最后一个元素相同，若相同，则继续向后判断，若不同，
        则插入前面非重复有序表的最后，直到循环结束;
3°：结束后，返回1表示成功的标志;
*****************************************************************************************
*/
        //index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
        //               0,1,2,3,3,3,4,5,5,6,7, 8, 9, 9, 10

int DeleteRepatElem(sqList& L) {
        if (L.length == 0) {
                return -1;
        }
        int i, j;   //i用来存储第一个不相同的元素，j是遍历指针
        for (i = 0, j = 1; j < L.length; j++) {
                if (L.data[i] != L.data[j]) {  //查找下一个与上个元素值不同的元素
                        L.data[++i] = L.data[j];   //找到后，将元素前移
                }
        }
```

```
        L.length = i + 1;
        return 1;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.2 , 应用题第7题 (P18)
*********************************
1、题目:
        将两个有序顺序表合并成一个新的有序顺序表, 并由函数返回结果顺序表。
        (Combine the two ordered sequence tables into a new ordered sequence table,
        and return the result sequence table by the function.)

2、算法思想:
1°: 若传入的线性表为空, 则返回-1, 表示失败标志;
2°:  首先, 按顺序不断取下两个顺序表表头较小的结点存入新的顺序表中。然后, 看哪个表还有剩余,
将剩下的部分加到新的顺序表后面。;
3°: 结束后, 返回1表示成功的标志;
*****************************************************************************************
*/

int MergeList(sqList L1, sqList L2, sqList &L) {
        L.length = 0;
        int i=0, j=0, k=0;
        int n = L1.length + L2.length;

        while (i < L1.length && j < L2.length) {
                if (L1.data[i] <= L2.data[j]) {
                        L.data[k++] = L1.data[i++];
                }else {
                        L.data[k++] = L2.data[j++];
                }
        }
        while (i < L1.length) {
                L.data[k++] = L1.data[i++];
```

```
        }
        while (j < L2.length) {
                L.data[k++] = L2.data[j++];
        }
        L.length = k;
        return 1;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.2 , 应用题第8题 (P18)
*********************************
1、题目:
        已知在一维数组A[m+n]中依次存放着两个线性表 (a1,a2,a3,....,am) 和 (b1,b2,b3,....,bn) ;
试编写一个函数,
        将数组中两个顺序表的位置互换, 即将b1,b2,b3,....,bn) 放在 (a1,a2,a3,....,am) 的前面。
        (Combine the two ordered sequence tables into a new ordered sequence table,
        and return the result sequence table by the function.)

2、算法思想:
1°: 若传入的线性表为空, 则返回-1, 表示失败标志;
2°: 首先将线性表[m+n]中的全部元素 (a1,a2,a3,....,am,b1,b2,b3,....,bn) 原地逆置为
         (bn,bn-1,bn-2,...,b1,am,am-1,am-2,....,a1) , 再对前n个元素和后m个元素分别使用逆置算
法,
        就可以得到 (b1,b2,b3,....,bn,a1,a2,a3,....,am) , 从而实现顺序表的位置互换。
3°: 结束后, 返回1表示成功的标志;
*********************************************************************************************
*/

int SwapSeat(sqList& L, int m) {
        if (L.length == 0 || m < 0) {
                return -1;
        }
        int temp;
        for (int i = 0, j = L.length-1; i < j; i++) {
                temp = L.data[i];
                L.data[i] = L.data[j];
                L.data[j--] = temp;
        }
```

```
        for (int i = 0, j = L.length - 1 - m; i <= j; i++) {
                temp = L.data[i];
                L.data[i] = L.data[j];
                L.data[j--] = temp;
        }
        for (int i = L.length - m, j = L.length - 1; i <= j; i++) {
                temp = L.data[i];
                L.data[i] = L.data[j];
                L.data[j--] = temp;
        }
        return 1;
}
```

```
#include "stdafx.h"
void insertX(sqList& L, int index, int x);
/********************王道2019年chapter-2.2 ，应用题第9题 (P18)
*********************************
```

1、题目:

线性表 (a1,a2,a3,....,an) 中元素递增有序且按顺序存储于计算机内。要求设计一算法完成用最少时间在表中查找数值为x的元素，

若找到将其与后继元素位置相交换，若找不到将其插入表中并使表中元素仍递增有序。

2、算法思想:

1°：若传入的线性表为空或者传入的操作数x正好是线性表最后一位数据元素，则返回-1，表示失败标志;

2°： 。

3°：结束后，返回1表示成功的标志;

```
**************************************************************************************************
*/



int OperationX(sqList &L, int x) {
	if (L.length == 0 || x == L.data[L.length - 1]) {
		return -1;
	}
	int i, temp, sub;
	int j = 0, nearX = abs(L.data[0] - x);
	for (i = 0; i < L.length; i++) {

		//若能找到x，则与其后继元素进行位置交换
		if (L.data[i] == x) {
			temp = L.data[i];
			L.data[i] = L.data[i + 1];
			L.data[i + 1] = temp;
```

```
                return 1;
        }

        //若x不在线性表里，则找到它该插入的位置
        sub = abs(x - L.data[i]);
        if (sub !=0 && sub < nearX) {
                nearX = sub;
                j = i;
        }
}

//说明没有这个数，那么，执行在此位置的插入操作,将j本身及之后所有的数依次后移
L.length += 1;
for (int i = L.length - 1; i > j; i--) {
        L.data[i] = L.data[i - 1];
}
if (x > L.data[j]) {
        L.data[j + 1] = x;
}else {
        L.data[j] = x;
}
return 1;
}
```

```
#include "stdafx.h"


//1-1、创建一个单向链表的节点
LNode* createNode(int data) {
	LNode* newLNode = (LNode*)malloc(sizeof(LNode));
	if (newLNode){
		newLNode->data = data;
		newLNode->next = NULL;
	}
	return newLNode;
}



//1-2、创建一个双向链表的结点
DNode* createDNode(int data) {
	DNode *newDNode = (DNode*)malloc(sizeof(DNode));
	if (newDNode){
		newDNode->prior = newDNode->next = NULL;
		newDNode->data = data;
	}
	return newDNode;
}



//2-1、创建一个带头结点的单向链式表*/
LinkList createLinkListH() {
	LinkList headNode = (LinkList)malloc(sizeof(LinkList));
```

```
        if (headNode) {
                headNode->next = NULL;
        }
        return headNode;
}


//2-2、创建一个不带头结点的单向链式表
LinkList** createLinkList() {
        LinkList **firstNode = (LinkList**)malloc(sizeof(LinkList**));
        if (firstNode){
                **firstNode = NULL;
        }
        return firstNode;
}


//2-3、创建一个带头结点的双向链表
DLinkList createDLinkListH() {
        DLinkList headNode = (DLinkList)malloc(sizeof(DNode));
        if (headNode){
                headNode->next = headNode->prior = NULL;
        }
        return headNode;
};


//2-4、创建一个不带头结点的双向链表
DLinkList* createDLinkList() {
```

```
        DLinkList *headNode = (DLinkList*)malloc(sizeof(DLinkList*));
        if (headNode) {
                headNode = NULL;
        }
        return headNode;
}



//3-1、头插法向单链表里插入数据
void insertLNodeByHead(LinkList &L, int data) {
        LNode *newLNode = createNode(data);
        newLNode->next = L->next;
        newLNode->data = data;
        L->next = newLNode;
}



//3-2、尾插法向单链表里插入数据
void insertLNodeByTail(LinkList &L, int data){
        LNode *newNode = createNode(data);
        LNode *tailNode = L;
        while (tailNode->next) {
                tailNode = tailNode->next;
        }
        tailNode->next = newNode;
}


//3-3、头插法向双链表里插入数据
```

```
void insertDNodeByHead(DLinkList &DL, int data) {
        DNode *newNode = createDNode(data);
        newNode->next = DL->next;
        if (DL->next != NULL) {
                DL->next->prior = newNode;
        }
        DL->next = newNode;
        newNode->prior = DL;
}


//3-4、尾插法向双链表里插入数据
void insertDNodeByTail(DLinkList& DL, int data) {
        DNode* newNode = createDNode(data);
        DNode* tailNode = DL;
        while (tailNode->next){
                tailNode = tailNode->next;
        }
        tailNode->next = newNode;
        newNode->prior = tailNode;
}




//4-1、从头至尾打印单链表
void printLinkList(LinkList L) {
        LNode* p = L->next;
        if (p) {
                while (p->next) {
```

```
                printf("%d-->", p->data);
                p = p->next;
            }
            printf("%d\n", p->data);
        }else {
            printf("NULL\n");
        }

}


//4-1-1、从头至尾打印循环单链表
void printCycleLinkList(LinkList L) {
        LNode* p = L->next;
        if (p) {
            while (p->next != L) {
                printf("%d-->", p->data);
                p = p->next;
            }
            printf("%d\n", p->data);
        }
        else {
            printf("NULL\n");
        }
}
```

```
//4-2、从头至尾打印双链表
void printDLinkList(DLinkList DL) {
        DNode* p = DL->next;
        while (p->next) {
                printf("%d-->", p->data);
                p = p->next;
        }
        printf("%d\n", p->data);
}

//4-3、从尾至头部打印双链表
void printDlinkListInvert(DLinkList DL) {
        DNode* p = DL->next;
        while (p->next){
                p = p->next;
        }
        while (p->prior != DL){
                printf("%d-->", p->data);
                p = p->prior;
        }
        printf("%d\n",p->data);
}



//5-1、求一个单链表的长度
int getLinkLength(LinkList L) {
        LNode* p = L->next;
        int len = 0;
        while (p) {
                p = p->next;
```

```
                len++;
        }
        return len;
}
```

```
//6-1、定位一个元素的位置，在链表中，返回一个组织向它的指针(题2.3.20专用)
DNode1* LocateElem(DLinkList1 DL, int data) {
        DNode1* p = DL->next;
        while (p && p->data != data) {
                p = p->next;
        }
        return p;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第1题（P37）
*********************************
1、题目：
        设计一个递归算法，删除不带头结点的单链表L中所有值为x的结点；
2、算法思想：
1°：设f(L, x) 的功能是删除以L为首结点指针的单链表中所有值等于x的结点，则显然有f（L->next,
x) 的功能是删除以L->next为首结点指针的单链表中所有值等于x的结点。
        由此，可以推出递归模型如下；
2°：终止条件：f（L, x) =不做任何事情；                    若L为空表；
3°：递归主体：f（L, x) =删除*L结点；fL->next, x) ；     若L->data==x
                        f（L, x) =f（L->next, x) ；              其他情况；
*************************************************************************************
*/

void DeleteXOfRecursion(LinkList &L, int x) {
        LNode* p;
        if (L ==  NULL) {
                return;
        }
        if (L->data == x) {
                p = L;
                L = L->next;
                free(p);
                DeleteXOfRecursion(L, x);
        }else{
                DeleteXOfRecursion(L->next,x);
        }
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第2题（P37）
*********************************
1、题目:
        在带头结点的单链表L中，删除所有值为x的结点，并释放其空间，假设值为x的结点不唯一，试编写算法以实现上述操作。
2、算法思想:  [Function 1]
1°：用p从头至尾扫描单链表，pre指向p结点的前驱;
2°：若p所指结点的值为x，则删除，并让p移向下一个结点;
3°：否则让pre、p指针同步后移一个结点。;
*****************************************************************************************
*/
void DeleteX_1(LinkList& L, int x) {
        LNode* pre  = L;
        LNode* p = L->next;
        LNode* q;
        while (p) {
                if (p->data == x) {
                        q = p;
                        p = p->next;
                        pre->next = p;
                        free(q);
                }else {
                        pre = p;
                        p = p->next;
                }
        }
}
```

/**********王道2019年chapter-2.3 ，应用题第2题（P37）

****

1、题目:

在带头结点的单链表L中，删除所有值为x的结点，并释放其空间，假设值为x的结点不唯一，试编写算法以实现上述操作。

2、算法思想: [Function 2]

1°：采用尾插法建立单链表;

2°：用p指针扫描L的所有结点，当其值个为x时将其链按到L之后，否则将其释放;

****

*/

```
void DeleteX_2(LinkList& L, int x) {
        LNode* p = L->next;
        LNode* r = L;
        LNode* q;
        while (p) {
                if (p->data != x) {
                        r->next = p;
                        r = p;
                        p = p->next;
                }else {
                        q = p;
                        p = p->next;
                        free(q);
                }
        }
        r->next = NULL;
}
```

```
#include "stdafx.h"

/*******************王道2019年chapter-2.3 ，应用题第3题（P37）
*********************************
1、题目:
        设L为带头结点的单链表，编写算法实现从尾到头反向输出每个结点的值。
2、算法思想:  [Function 1]
1°：因为单链表要想逆向输出，可以使用一个栈，也可以使用尾插法来再次新建一个链表，也可以用递归四思想，此处用递归思路;
2°：若当前节点的next不为空，则继续递归进入链表L->next;
3°：递归出口是当前节点的next域为空时，即逐层输出当前节点data域;
*****************************************************************************************
*/

void ReverseOutput(LinkList L) {
        if (L->next) {
                ReverseOutput(L->next);
        }
        printf("%d-->", L->data);
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第4题（P37）
*********************************
1、题目:
        试编写在带头结点的单链表L中删除一个最小值结点的高效算法（假设最小值结点是唯一的）
2、算法思想:  [Function 1]
1°：新建4个遍历指针分别为，pre:始终指向p的前驱； minPre:始终指向min的前驱；p是遍历指针，min用来保存最小值节点;
2°：如果p->data < minP->data，则将p赋给minP，同时将pre赋给minPre；如果不小于，则将pre和p依次后移;
3°：删除minP;
*****************************************************************************************
*/

void DeleteMin(LinkList& L) {
        LNode* pre = L, *p = pre->next;
        LNode* minPre = pre, *minP = p;
        while (p) {
                if (p->data < minP->data) {
                        minP = p;
                        minPre = pre;
                }
                pre = p;
                p = p->next;
        }
        minPre->next = minP->next;
        free(minP);
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-2.3 ，应用题第5题（P37）
*********************************
1、题目:
        试编写算法将带头结点的单链表就地逆置，所谓“就地”是指辅助空间复杂度为0(1)
2、算法思想:
1°：将头结点摘下，然后从第一结点开始，依次前插入到头结点的后面（头插法建立单链表），直到最后一个结点为止，则实现了链表的逆置;
*****************************************************************************************
*/

void InvertLink(LinkList &L) {
        LNode* p, * r;
        p = L->next;
        L->next = NULL;  //将头节点摘下
        while (p) {
                r = p->next;    //防止断连，用r来暂存p
                p->next = L->next;    //将p节点插入到头节点之后
                L->next = p;
                p = r;
        }
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-2.3 ，应用题第6题（P37）
*********************************
1、题目:
        有一个带头结点的单链表L，设计一个算法使其元素递增有序。
2、算法思想:
        1°：算法思想：采用直接插入排序算法的思想，先构成只含一个数据结点的有序单链表;
        2°：然后依次扫描单链表中剩下的结点*p（直至p==NULL为止）;
        3°：在有序表中通过比较查找插入*p的前驱结点*pre，然后将*p插入到*pre之后。
*************************************************************************************
*/

void AscendingOrder(LinkList& L) {
        LNode* p = L->next, * pre;
        LNode* r = p->next;  //r保存*P后继结点指针，以保证不断链
        p->next = NULL;      // 构造只含一个数据结点的有序表
        p = r;
        while (p) {
                r = p->next;     // 保存 * p的后继结点指针
                pre = L;
                while (pre->next != NULL && pre->next->data < p->data) {
                        pre = pre->next;     //在有序表中查找插入*p的前驱结点*pre
                }
                p->next = pre->next;     //将*p插入到*pre之后
                pre->next = p;           //扫描原单链表中剩下的节点
                p = r;
        }
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第7题（P37）
*********************************
```

1、题目：

设在一个带表头结点的单链表中所有元素结点的数据值无序，试编写一个函数，删除表中所有介于给定的两个值（作为函数参数给出）之间的元素的元素（若存在）。

2、算法思想：

1°：定义两个指针*pre 和 *p，p用来遍历单链表，pre用来指向其前驱节点；

2°：依次遍历，找到符合条件的将其删除，否则，p = p->next,继续遍历

```
**********************************************************************************************
*/

void DeleteS_E(LinkList& L, int start, int end) {
	if (start > end) {
		return;
	}
	LNode* pre = L;
	LNode* p = pre->next;
	while (p) {
		if (start < p->data && p->data < end) {
			pre->next = p->next;
			free(p);
			p = pre->next;
		}else {
			pre = p;
			p = p->next;
		}
	}
}
```

```
#include "stdafx.h"
int getLinkLen(LinkList L);
/********************王道2019年chapter-2.3 ，应用题第8题（P37）
*********************************
```

1、题目：

给定两个单链表，编写算法找出两个链表的公共结点。

2、算法思想：

0°：核心思想：两个单链表有公共结点，也就是说两个链表从某一结点开始，它们的next都指向同一个结点。由于每个单链表结点只有一个next域，

因此从第一个公共结点开始，之后它们所有的结点都是重合的，不可能再出现分叉。所以，两个有公共结点而部分重合的单链表，

拓扑形状看起来像Y，而不可能像X。

1°：先求出两个单链表的长度，假设他们长度差值为k,然后定义两个遍历指针P1和P2,找出他们长度较大的那个单链表，将其向后移动k个位置，以达到尾部对齐；

2°：然后判断两个遍历指针的next是否相等，若相等，则返回这个节点及其之后的所有节点为其他们的公共节点

```
*********************************************************************************************
*/

//方法（一）
LinkList FindPublicNode_1(LinkList L1, LinkList L2) {
	if (!(L1->next && L2->next)) {
		return NULL;			//判空
	}
	int lenL1 = getLinkLen(L1);  //求出啊L1和L2的长度
	int lenL2 = getLinkLen(L2);
	LNode* p1 = L1->next;		//初始化遍历指针P1和P2
	LNode* p2 = L2->next;
	LinkList L;					//用来保存最终的公共链表
```

```
        int k, j = 0;                   //k用来记录L1和L2的长度的差值，j用来记录长度较长的指针向后移动的个数
        if (lenL1 >= lenL2) {
                k = lenL1 - lenL2;
                while (j++ != k) {
                        p1 = p1->next;
                }
                while (p2->next != p1->next) {
                        p1 = p1->next;
                        p2 = p2->next;
                }
                L =  (LinkList)p2;
        }else {
                k = lenL2 - lenL1;
                while (j++ != k) {
                        p2 = p2->next;
                }
                while (p2->next != p1->next) {
                        p1 = p1->next;
                        p2 = p2->next;
                }
                L = (LinkList)p1;
        }
        printf_s("k==%d", k);
        return L;
}
```

//方法（二）

```
LinkList FindPublicNode_2(LinkList L1, LinkList L2) {
        int len1 = getLinkLen(L1);
        int len2 = getLinkLen(L2);
        int dist;
        LinkList shortList, longList;
        if (len1 > len2) {
                longList = L1->next;
                shortList = L2->next;
                dist = len1 - len2;
        }else {
                longList = L2->next;
                shortList = L1->next;
                dist = len2 - len1;
        }

        while (dist--){
                longList = longList->next;
        }

        while (longList != NULL){
                if (longList->next == shortList->next) {
                        return longList;
                }else{
                        longList = longList->next;
                        shortList = shortList->next;
                }
        }
        return NULL;
}
```

```
//求单链表的长度
int getLinkLen(LinkList L) {
        LNode* p = L->next;
        int len = 0;
        while (p) {
                p = p->next;
                len++;
        }
        return len;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第9题（P37）
*********************************
```

1、题目：

给定一个带表头结点的单链表，设head为头指针，结点的结构为（data，next），data为整型元素，next为指针，

试写出算法：按递增次序输出单链表中各结点的数据元素，并释放结点所占的存储空间（要求：不允许使用数组作为辅助空间）。

2、算法思想：

方法(一)：

遍历整个单链表，从头开始，两辆依次比较，如果是小于，则向后移动遍历指针，一次一趟下去，则会找到最小节点，将其删除即可；

方法(二)：

县将单链表排序，然后顺序输出，同时，依次删除

```
*********************************************************************************************
*/

void IncrementPrintLink_1(LinkList& L) {
	LNode *pre, *p, *u;
	while (L->next) {
		pre = L;
		p = pre->next;
		while (p->next) {
			if (p->next->data < pre->next->data) {
				pre = p;
			}
			p = p->next;
		}
		if (L->next->next) {
```

```
                printf_s("%d-->", pre->next->data);
                u = pre->next;
                pre->next = u->next;
                free(u);
            }else {
                printf_s("%d\n", pre->next->data);
                u = pre->next;
                pre->next = u->next;
                free(u);
            }

        }
        L->next = NULL;
}


/*  先将单链表里的数据递增排序，排好序后在输出，最后删除这个链表*/
void IncrementPrintLink_2(LinkList& L) {
        LNode* p = L->next;
        LNode* pre;
        LNode* r = p->next;
        p->next = NULL;
        p = r;
        while (p) {
            r = p->next;
            pre = L;
            while (pre->next != NULL && pre->next->data < p->data) {
                pre = pre->next;
            }
            p->next = pre->next;
```

```
            pre->next = p;
            p = r;
        }
        LNode* qre = L;
        LNode* q = qre->next;
        LNode* u;
        while (q->next) {
            printf_s("%d-->", q->data);
            u = qre->next;
            qre->next = u->next;
            q = q->next;
            free(u);
        }
        printf_s("%d\n", q->data);
        L->next = NULL;
        //free(L);
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-2.3 ，应用题第10题（P37）
*********************************
```

1、题目：

将一个带头结点的单链表A分解为两个带头结点的单链表A和B，使得A表中含有原表中序号为奇数的元素，

而B表中含有原表中序号为偶数的元素，且保持其相对顺序不变。

2、算法思想：

1°：设置一个访问序号变量（初值为0），每访问一个结点序号自动加1，然后根据序号的奇偶性将结点插入到A表或B表中。重复以上操作直到表尾。

3、拓展延申：

为了保持原来结点中的顺序，本题采用尾插法建立单链表。此外，本算法完全可以不用设置序号变量。while循环中的代码改为将结点插入到表A中和将下一结点插入到表B中，

这样while中第一处理的结点就是奇数号结点，第二处理的结点就是偶数号结点。

```
*************************************************************************************************
*/


/*
算法思想：
设置一个计数变量i，每访问一个节点使其自增，根据i的奇偶性将当前节点分别连到不同的链表上；
通过尾插法的方法将对应节点连接成两个链表。
*/
void SplitLink_1(LinkList &A, LinkList &B) {
        int i = 0;                  //计数变量
        LNode *ra = A, *rb = B;    //尾插法插入，尾指针
        LNode *p = A->next;        //遍历指针
        A->next = NULL;            //置空A链表
```

```
        while (p) {
                i++;
                if (i % 2 == 0) {
                        rb->next = p;
                        rb = p;
                }else {
                        ra->next = p;
                        ra = p;
                }
                p = p->next;
        }
        ra->next = NULL;
        rb->next = NULL;
}
```

```
/**
 * 算法思想:
                本算法完全可以不用设置序号变量。
                while循环中的代码改为将结点插入到表A中和将下一结点插入到表B中,
                这样while中第一处理的结点就是奇数号结点, 第二处理的结点就是偶数号结点。
 */
void SplitLink_2(LinkList& A, LinkList& B) {
        LNode *ra = A, *rb = B;
        LNode *p = A->next;
        A->next = NULL;
        while (p) {
                ra->next = p;
                ra = p;
```

```
            p = p->next;
            rb->next = p;
            rb = p;
            if (!p) break;
            p = p->next;
        }
        ra->next = NULL;
        rb->next - NULL;
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-2.3 ，应用题第11题（P37）
*********************************
1、题目:
        设C={a1，b1，a2，b2，...，an，bn}为线性表，采用带头结点的hc单链表存放，设计一个就地算法，将其拆分为两个线性表，使得
        A={a1，a2，...，an}，B={bn，...，b2，b1}。
2、算法思想:
        1°：用P指针来遍历整个链表，若是奇数序号元素则尾插到A链表中，偶数序号则头插到B链表中

*****************************************************************************************
*/

/*方法1*/
LinkList splitIntoAB_1(LinkList& A) {
        LinkList B = (LinkList)malloc(sizeof(LNode));
        if (B) B->next = NULL;                                          //初始化B链表为空链表

        LNode* p = A->next, * q;                                        //p 是遍历指针，q用来临时保存在头插法时的p指针
        LNode* ra = A;                                                  //ra 是A链表的尾指针，
        A->next = NULL;
        int cursor = 0;                                                 //游标，用来标记当前节点是偶数号或奇数号
        while (p) {
                cursor++;
                if (cursor % 2 == 1) {                                  //奇数号节点，则尾插到A链表
                        ra->next = p;
                        ra = p;
```

```
                p = p->next;
            }
            else {                                          //偶数号节点，则头插到B链表
                q = p->next;                                //一定先保存下p指针的状态，此时，P = p.next，while循环外无需再将p向后移动
                p->next = B->next;
                B->next = p;                                //将上面用q保存下来的p的状态再还给p
                p = q;
            }
        }
        ra->next = NULL;
        return B;
}



/*方法2*/
LinkList splitIntoAB_2(LinkList& A) {
        LinkList B = (LinkList)malloc(sizeof(LNode));
        if (B) B->next = NULL;                              //初始化B链表为空链表

        LNode* p = A->next, * q;                            //p 是遍历指针，q用来临时保存在头插法时的p指针
        LNode* ra = A;                                      //ra 是A链表的尾指针，
        A->next = NULL;
        while (p) {
            ra->next = p;                                   //尾插法向A插入节点
            ra = p;
```

```
            p = p->next;                    //插完后，向后移动p指针，此时到偶数号节点，则头插到B链表
            if (!p) break;                  //
            q = p->next;                    //一定先保存下p指针的状态，此时，P = p.next，while循环外无需再将p向后移动
            p->next = B->next;
            B->next = p;                    //将上面用q保存下来的p的状态再还给p
            p = q;

        }
        ra->next = NULL;
        return B;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第12题（P37）
*********************************
1、题目：
        在一个递增有序的线性表中，有数值相同的元素存在。若存储方式为单链表，设计算法去掉数值相同的元素，使表中不再有重复的元素。
        例如（7，10，10，21，30，42，42，42，51，70）将变作（7，10，21，30，42，51，70）。
2、算法思想：
        1°：因为是递增有序的链表，即重复的元素一定是连续出现的，所以用一个pre(始终指向p的前驱)和p指针来遍历整个链表；
        2°：若发现pre和p的data域相同时，则删除p的结点，同时保持pre指向不动，p继续后移；
        3°：若p和pre的data域不同时，则将pre和p同时向后移动，直到链表的最后，表中自然就剩下了所有的非重复元素。
*********************************************************************************************
*/
void deleteRepetitionElem(LinkList &L) {
        if (!(L && L->next && L->next->next) ) {
                printf("链表中只有一个元素或链表为空!!!\n");
                return;
        }
        LNode* pre = L, *p = L->next;
        while (p) {
                if (pre->data == p->data) {                             //若遇到重复元素，则删除后面的重复元素
                        pre->next = p->next;
                        //pre = p;
                        p = p->next;
                }else {
//不是重复元素，则p和pre同时向后遍历
                        pre = p;
```

```
                p = p->next;
            }
        }
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第13题 (P37)
*********************************
1、题目:
        假设有两个按元素值递增次序排列的线性表，均以单链表形式存储。请编写算法将这两个单链表归并为一个按元素值递减次序排列的单链表，
        并要求利用原来两个单链表的结点存放归并后的单链表。
2、算法思想:
        1°：用两个指针pa 和 pb来分别遍历A 和 B 链表;
        2°：因为A和B已是递增有序的，将A和B同时向后遍历;
        3°：若pa的值小于pb的值，则将pa头插到C链表中，同时将pa向后移动一个位置，pb保持不动;
        4°：若pa的值大于pb的值，则将pb头插到C链表中，同时将pb向后移动一个位置，pa保持不动;
        5°：若最后pa或pb遍历的链表还剩下元素，则将其头插到C链表即可。
*************************************************************************************
*/
void mergeABToAB(LinkList &A, LinkList &B) {
        LNode *pa = A->next,  *pb = B->next;
        A->next = NULL;
        LNode *q;
        while (pa && pb) {
                if (pa->data < pb->data) {                      //若pa的值小于pb的值，则将pa头插到C链表中，同时将pa向后移动一个位置，pb保持不动
                        q = pa->next;                                                   //关键操作，必须要用临时指针q来保存当前遍历指针的后继节点，保证不断链
                        pa->next = A->next;
                        A->next = pa;
                        pa = q;
                }else {                                         //若pa的值大于pb的值，则将pb头插到C链表中，同时将pb向后移动一个位置，pa保持不动
                        q = pb->next;
```

```
                pb->next = A->next;
                A->next = pb;
                pb = q;
            }
        }

        //若pa不为空
        if (pa) pb = pa;
        while (pb) {
            q = pb->next;
            pb->next = A->next;
            A->next = pb;
            pb = q;
        }
        free(pb);
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第14题（P37）
*********************************
```

1、题目：

设A和B是两个单链表（带头结点），其中元素递增有序。设计一个算法从A和B中公共元素产生单链表C，要求不破坏A、B的结点。

2、算法思想：

1°：分别为A、B链表设置两个遍历指针pa和pb，两个链表从头开始同时向后遍历；

2°：若pa.data<pb.data，则将pa向后移动，若pb.data<pa.data,则将pb向后移动；

3°：只有在pa.data和pb.data相等时，动态创建一个新节点q，然后将当前pa或pb的值赋给一个q，并将节点q尾插到C链表上。

```
*************************************************************************************
*/
LinkList getPublicOfAB(LinkList A, LinkList B) {
        LNode *pa = A->next,  *pb = B->next,  *q,  *r;                          //pa 和 pb 是工作指针，r是C链表的尾指针，q是一个临时指针
        LinkList C = (LNode*)malloc(sizeof(LNode));
        C->next = NULL;                                                         //初始化一个带头结点的单链表C
        r = C;
        while (pa && pb) {
                if (pa->data < pb->data) {                                      //若pa.data<pb.data，则将pa向后移动
                        pa = pa->next;
                } else if (pa->data > pb->data) {                               //若pb.data<pa.data,则将pb向后移动
                        pb = pb->next;
                } else {                                                        //此时，pa.data == pb.data,进行新结点的创建和连接操作
```

```
                q = (LNode*)malloc(sizeof(LNode));
                q->data = pa->data;
                r->next = q;
                r = q;
                pa = pa->next;
                pb = pb->next;
            }
        }
        r->next = NULL;
        return C;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第15题 (P37)
*********************************
```

1、题目:

已知两个链表A和B分别表示两个集合，其元素递增排列。编制函数，求A与B的交集，并存放于A链表中。

2、算法思想:

1°：定义两个指针pa 和 pb 来扫描A链表和B链表;

2°：若在扫描时发现他们所指向的data域值相等，则将其尾插到A链表里，然后将pa和pb同时后移;

3°：若pa.data < pb.data ,则将pa向后移动，pb位置保持不变，若pa.data > pb.data ,则将pb向后移动，pa位置保持不变;

4°：遍历结束后，若有其中一个链表内有剩余的元素，则将他全部删除即可

```
*********************************************************************************************
*/
LinkList getIntersectionOfAB(LinkList &A, LinkList &B) {

        LNode *pa = A->next,  *pb = B->next;            //遍历指针pa 和 pb
        LNode *ra = A;                                  //A 链表的尾指针
        LNode *u;                                       //用来释放指针指向的节点

        while (pa && pb) {
                if (pa->data == pb->data) {                 //pa 和pb指向的data相等时，将当前节点尾插到A链表，并释放这两个节点
                        ra->next = pa;
                        ra = pa;
                        pa = pa->next;
                        u = pb;
                        pb = pb->next;
                        free(u);
```

```
            }else if (pa->data < pb->data) {                    //将pa后移一位，pb保持位置不变,并释放pa指向的节点
                    u = pa;
                    pa = pa->next;
                    free(u);
            }else {                     //将pb后移一位，pa保持位置不变,并释放pb指向的节点
                    u = pb;
                    pb = pb->next;
                    free(u);
            }
    }

    if(pa) pb = pa;                                     //若条件成立，则说明pa中有剩余节点，pb已经是空的；若不成立，则反之

    while (pb) {
            u = pb;
            pb = pb->next;
            free(u);
    }
    ra->next = NULL;
    return A;
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-2.3 ，应用题第16题 (P38)
*********************************
1、题目:
        两个整数序列A=a1，a2，a3，... ，am和B=b1，b2，b3，... ，bn已经存入两个单链表中，设计一个算法，判断序列B是否是序列A的连续子序列。
2、算法思想:
        1°：定义两个遍历指针pa和pb分别遍历A、B两个链表，一个指针pre来时刻记录每次pa指针的前驱位置;
        2°：若他们的data域相等，则两个同时向后移动，继续遍历;
        3°：若他们的data域不相等，则将pa指针通过pre的记录，将pa回溯到pre.next位置后继续遍历，将pb指针回溯到B链表初始位置。
*************************************************************************************
*/
bool isSubsequence(LinkList A, LinkList B) {

        LNode *pa = A->next, *pb = B->next;
        LNode *pre = pa;                                        //pre时刻记录着pa指针在A链表中遍历时的节点前驱

        while (pa && pb) {
                if (pa->data == pb->data) {                     //若他们的data域相等，则两个同时向后移动，继续遍历
                        pa = pa->next;
                        pb = pb->next;
                } else {                                        //若他们的data域不相等;
                        pre = pre->next;                        //则将pa指针回溯到上面进入if之前的位置的next位置;
                        pa = pre;
```

```
                pb = B->next;                                   //同时将pb指针回到B
```
链表的初始位置两个同时向后移动，继续遍历。
```
                }
        }


        if (pb) {
                return false;                                   //若在循环结束后pb没有
```
遍历到B链表的最后一个位置，则说明B链表不是A链表的子序列
```
        } else {
                return true;
        }
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-2.3 ，应用题第17题（P38）
*********************************
1、题目：
        设计一个算法用于判断带头结点的循环双链表是否对称。
2、算法思想：
        1°：定义两个指针p、pri，初始时分别指向双循环链表的next节点和prior节点；
        2°：若pri指针的data域和p->nex的data域相等，则将pri继续向前移动一个，p继续向后移动一个；若不相等，则不满足中心对称，直接返回false；
        3°：因为是循环链表，所以判断一趟遍历完成的条件是： pri != p && pri->next != p
*****************************************************************************************
*/
bool isSymmetryDLink(CDLinkList CDL) {
        CDNode *pri = CDL->prior;
        CDNode *p = CDL->next;
        while (pri != p && pri->next != p) {
                if (pri->data == p->data) {
                        pri = pri->prior;
                        p = p->next;
                } else {
                        return false;
                }
        }
        return true;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第18题 (P38)
*********************************
1、题目:
        有两个循环单链表，链表头指针分别为h1和h2，编写一个函数将链表h2链接到链表h1之后，要求链接后的链表仍保持循环链表形式。
2、算法思想:
        1°：找到A链表和B链表的尾指针;
        2°：将B链表的首元素连接在A链表之后;
*********************************************************************************************
*/
void jointAB(LinkList &A, LinkList &B) {
        LNode *pa = A,  *pb = B;
        while (pa->next != A) {
                pa = pa->next;
        }

        while (pb->next != B) {
                pb = pb->next;
        }
        pa->next = B->next;
        pb->next = A;
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-2.3 ，应用题第19题 (P38)
*********************************
```

1、题目:

设有一个带头结点的循环单链表，其结点值均为正整数。设计一个算法，反复找出单链表中结点值最小的结点并输出，

然后将该结点从中删除，直到单链表空为止，再删除表头结点。

2、算法思想:

1°：定义pre和p指针分别为遍历指针的前驱和遍历指针；定义minpre和minp指针为当前遍历出的节点中的最小值的前驱和最小值节点

2°：找到最小值节点后将其删除即可

```
*********************************************************************************************
*/
void asecdingOutPut(LinkList &L) {
	LNode *pre, *p, *minpre, *minp;
	while (L->next != L) {
		p = L->next;
		pre = L;
		minp = p;
		minpre = pre;
		while (p != L) {
			if (p->data < minp->data) {
				minp = p;
				minpre = pre;
			}
			pre = p;
			p = p->next;
		}
		printf("%d-->", minp->data);
		minpre->next = minp->next;
```

```
            free(minp);
        }
        printf("\n");
}
```

```
#include "stdafx.h"

/*******************王道2019年chapter-2.3 ，应用题第20题（P38）
*********************************
1、题目:
        设头指针为L的带有表头结点的非循环双向链表，其每个结点中除有pred（前驱指针）、data（数据）和next（后继指针）域外，还有一个访问频度域freq。
        在链表被启用前，其值均初始化为零。每当在链表中进行一次Locate（L，x）运算时，令元素值为x的结点中freq域的值增1，
        并使此链表中结点保持按访问频度非增（递减）的顺序排列，同时最近访问的结点排在频度相同的结点的前面，以便使频繁访问的结点总是靠近表头。
        试编写符合上述要求的Locate（L，x）运算的算法，该运算为函数过程，返回找到结点的地址，类型为指针型。
2、算法思想:
        1°：
**********************************************************************************************
*/
DLinkList1 Locate(DLinkList1 &DL, int e) {




        return DL;
}
```

```
#include "stdafx.h"


//创建一个链式栈的节点
LSNode* createLSNode(int data) {
	LSNode* newLSNode = (LSNode*)malloc(sizeof(LSNode));
	if (newLSNode) {
		newLSNode->data = data;
		newLSNode->next = NULL;
	}
	return newLSNode;
}


//创建一个链式栈 (不带头结点)
LinkStack createLinkStack() {
	LinkStack firstLSNode = (LinkStack)malloc(sizeof(LinkStack));
	if (firstLSNode) {
		firstLSNode = NULL;
	}
	return firstLSNode;
}



//初始化一个顺序栈
void InitSqStatck(SqStack &S) {
	S.top = -1;
}

//初始化一个链式栈 (链式栈是不带头结点的)
```

```
void InitLinkStatck(LinkStack &LS) {
        LS = NULL;
}



//判断一个顺序栈是否为空
bool SqStackIsEmpty(SqStack S) {
        return S.top == -1;
}

//判断一个链式栈是否为空
bool LinkStackIsEmpty(LinkStack LS) {
        return LS == NULL;
}



//顺序栈的进栈操作
bool Push(SqStack &S, int x) {
        if (S.top == MaxSize - 1){
                return false;
        }
        S.data[++S.top] = x;
        return true;
}

//链式栈(不带头结点)的进栈操作
void Push(LinkStack &LS, int x) {
        LSNode *newLSNode = createLSNode(x);
        if (LS) {
```

```
                newLSNode->next = LS;
                LS = newLSNode;
        }else {
                LS = newLSNode;
        }
}



//顺序栈的出栈操作
bool Pop(SqStack &S, int &element) {
        if (S.top == -1) {
                return false;
        }
        element = S.data[S.top--];
        return true;
}

//链式栈的出栈操作 (int型)
bool Pop(LinkStack &LS, int &element) {
        if (!LS) {
                return false;
        }
        element = LS->data;
        LS = LS->next;
        return true;
}



//获取顺序栈顶元素
```

```
bool Top(SqStack S, int &element) {
        if (S.top == - 1) {
                return false;
        }
        element = S.data[S.top];
        return true;
}

//获取链式栈栈顶元素
bool Top(LinkStack LS, int &element) {
        if (!LS) {
                return false;
        }
        element = LS->data;
        return true;
}



//销毁顺序栈
void ClearSqStack(SqStack &S) {
        S.top = -1;
        //free(*S);
}

//销毁链式栈
void ClearLinkStack(LinkStack &LS) {
        LS = NULL;
        free(LS);
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-3.1 ，应用题第3题（P37）
*********************************
```

1、题目：

假设以I和O分别表示入栈和出栈操作。栈的初态和终态均为空，入栈和出栈的操作序列可表示为仅由I和O组成的序列，可以操作的序列称为合法序列，否则称为非法序列。

(1) 下面所示的序列中哪些是合法的？

A.IOIIOIOO　　B.IOOIOIIO

C.IIIOIOIO　　D.IIIOOIOO

(2) 通过对(1) 的分析，写出一个算法，判定所给的操作序列是否合法。若合法，返回true，否则返回false（假定被判定的操作序列已存入一维数组中）。

2、算法思想：

1°：用i作为遍历指针来遍历整个字符数组，遇到‘I’，sum加1，遇到‘O’，sum减1；

2°：sum表示当前节点及以前的所有操作的和，在减操作，也就是遇到‘O’时，要判断此时的sum是否小于0，若小于0，则表示序列不合法；

3°：最后判断sum的值，只有其值为0时，才是正确的序列，否则，即为不合法；

```
*********************************************************************************************
*/
bool JudgeSeriesIsTrue_1(char ch[]) {
        int sum = 0;
        int i = 0;
        while (ch[i] != '\0') {
                switch (ch[i]){
                 case 'I':
                         sum++;
                         break;
                 case 'O' :
                         sum--;
                         if (sum < 0) return false;
                         break;
```

```
                }
                i++;
        }
        if (sum == 0) {
                return true;
        }else {
                return false;
        }
}
```

/**********

****

2、算法思想:

1°：用i作为遍历指针来遍历整个字符数组，遇到‘I’，j表示遇到‘0’，k表示遇到‘I’；

2°：在减操作，也就是遇到‘0’时，要判断此时的j是否小于k，若大于k，则表示序列不合法；

3°：最后判断j和k的值，只有j==k时，才是正确的序列，否则，即为不合法；

****

**********/

```
bool JudgeSeriesIsTrue_2(char ch[]) {
        int i = 0;
        int j = 0;
        int k = 0;
        while (ch[i] != '\0') {
                switch (ch[i]) {
                        case 'I':
                                j++;
                                break;
                        case '0':
```

```
                        k++;
                        if (k > j) return false;
                        break;
                }
                i++;
        }
        if (j != k) {
                return false;
        }else {
                return true;
        }
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-3.1 ，应用题第4题 (P37)
*********************************
```

1、题目:

设单链表的表头指针为L，结点结构由data和next两个域构成，其中data域为字符型。

试设计算法判断该链表的全部n个字符是否中心对称。例如xyx、xyyx都是中心对称。

2、算法思想:

1°：使用栈来判断链表中的数据是否中心对称。将链表的前一半元素依次进栈;

2°：在处理链表的后一半元素时，当访问到链表的一个元素后，就从栈中弹出一个元素，两个元素比较，若相等，则将链表中下一个元素与栈中再弹出的元素比较，直至链表到尾;

3°：这时若栈是空栈，则得出链表中心对称的结论；否则，当链表中的一个元素与栈中弹出元素不等时，结论为链表非中心对称，结束算法的执行。

```
*********************************************************************************************
*/
bool IsCenterSymmetry(LinkStack LS, int n) {
        int i;                                                  //栈顶指针
        char a[MaxSize] = {0};                  //模拟栈(char 型)
        LSNode *p = LS;                                 //待检测序列的遍历指针
        char b = (char)0;                               //临时将int 型 转换为 char 型
        for (i = 0; i < n / 2; i++) {
                b = (char)p->data;                      //int 转 char
                a[i] = b;                                       //入栈
                p = p->next;
        }
        i--;
        if (n % 2 == 1) p = p->next;    //如果序列是奇数，则将p指针向后移动一格，以达到中心位置
        while (p != NULL && a[i] == (char)p->data) {
                i--;
                p = p->next;
```

```
        }
        if (i == -1) {
                return true;
        }else {
                return false;
        }
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-3.1 ，应用题第5题 (P37)
*********************************
1、题目:
        设有两个栈s1、s2都采用顺序栈方式，并且共享一个存储区[0，…，maxsize-1]，为了尽量利用空间，减少溢出的可能，
        可采用栈顶相向、迎面增长的存储方式。试设计s1、s2有关入栈和出栈的操作算法。
2、算法思想:
        1°：两个栈共享向量空间，将两个栈的栈底设在向量两端，初始时，s1栈顶指针为-1，s2栈顶指针为maxsize;
        2°：两个栈顶指针相邻时为栈满。两个栈顶相向、迎面增长，栈顶指针指向栈顶元素。
        3°：此处规定，S1的栈底为存储区[0]位置，S2的栈底为存储区[MaxSize-1]位置;
        4°：stk是一个共享栈
*****************************************************************************************
*/



/*****************************************************************************
入栈操作。i为栈号，i=0表示左边的s1栈，i=1表示右边的s2栈，x是入栈元素
入栈成功返回1，否则返回-1
*****************************************************************************/
bool StkPush(Stk &stk, int i, int x) {
        if (!(i==0 || i==1)) {
                printf_s("您数输入的栈号不对！！！\n");
                return false;
        }
        if (stk.top[1] - stk.top[0] == 1) {
                printf_s("栈已满！！！\n");
                return false;
```

```
        }
        switch (i){
                case 0:
                        stk.data[++stk.top[0]] = x;
                        return true;
                        break;
                case 1:
                        stk.data[--stk.top[1]] = x;
                        return true;
                        break;
        }
}
```

```
/*************************************************************************
退栈算法。i代表栈号，i=0时为s1栈，i=1时为s2栈
退栈成功返回退栈元素，否则返回-1
*************************************************************************/
bool StkPop(Stk& stk, int i, int &e) {
        if (!(i == 0 || i == 1)) {
                printf_s("您数输入的栈号不对！！！\n");
                return false;
        }

        switch (i){
                case 0:
                        if (stk.top[0] == -1) {
                                printf_s("S1栈已空！！！\n");
                                return false;
```

```
                }else {
                        e = stk.data[stk.top[0]--];
                        return true;
                }
        case 1:
                if (stk.top[1] == MaxSize) {
                        printf_s("S2栈已空！！！\n");
                        return false;
                }else {
                        e = stk.data[stk.top[1]++];
                        return true;
                }
        }

}
```

```
#include "stdafx.h"



//1、创建一个链式队列的节点
LNode* createLNode(ElemType data) {
	LNode* newLNode = (LNode*)malloc(sizeof(LNode));
	if (newLNode) {
		newLNode->data = data;
		newLNode->next = NULL;
	}
	return newLNode;
}


//1-1、创建一个链式队列 (包含着队头指针和队尾指针)
//在初始化里已经整合了


//2-1、初始化一个顺序队列
void InitSqQueue(SqQueue& SQ) {
	SQ.front = SQ.rear = 0;
}

/**
*2-2、初始化一个链式队列(带头结点的)
**/
void InitLinkQueue(LinkQueue &LQ) {
	LQ.front = LQ.rear = (LNode*)malloc(sizeof(LNode));
	if (LQ.front) {
		LQ.front->next = NULL;
```

```
        }

}



//3-1、判断顺序队列是否为空
bool isEmpty(SqQueue SQ) {
        if (SQ.front == SQ.rear) return true;
        else return false;
}

//3-2、判断链式队列是否为空
bool isEmpty(LinkQueue LQ) {
        if (LQ.front == LQ.rear) return true;
        else return false;
}



//4-1、顺序队列 (循环队列) 的进队
void EnQueue(SqQueue &SQ, ElemType x) {
        if ((SQ.rear + 1) % MaxSize == SQ.front) {
                printf("The Queue is full!!!\n");
                return;
        }
        SQ.data[SQ.rear] = x;
        SQ.rear = (SQ.rear + 1) % MaxSize;
}

//4-2、链式队列的进队(无需判断队满)
```

```
void EnQueue(LinkQueue &LQ, ElemType x) {
        LNode *newNode = createLNode(x);
        LQ.rear->next = newNode;
        LQ.rear = newNode;
}


//5-1、顺序队列 (循环队列) 的出队
void DeQueue(SqQueue &SQ, ElemType &e) {
        if (SQ.front == SQ.rear) {
                printf("The Queue is null!!!\n");
                return;
        }
        e = SQ.data[SQ.front];
        SQ.front = (SQ.front +1 ) % MaxSize;
}

//5-2、链式队列的出队 (需判断队空)
void DeQueue(LinkQueue &LQ, ElemType &e) {
        if (LQ.front == LQ.rear) {
                printf("The Queue is null!!!\n");
                return;
        }
        LNode *p = LQ.front->next;
        e = p->data;
        LQ.front->next = p->next;
        if (LQ.rear == p) LQ.rear = LQ.front;
        free(p);
}
```

```
//6-1、读取顺序队列队头元素
ElemType GetHead(SqQueue SQ) {
        if (!isEmpty(SQ)) {
                return SQ.data[SQ.front];
        } else {
                printf("The SqQueue is null in GetHead()!!!\n");
                return NULL;
        }
}

//6-2、读取循环队列的队头元素
ElemType GetHead(LinkQueue LQ) {
        if (!isEmpty(LQ)) {
                return LQ.front->next->data;;
        }else {
                printf("The LinkQueue is null in GetHead()!!!\n");
                return NULL;
        }
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-3.2 ，应用题第1题（P78）
*********************************
```

1、题目：

如果希望循环队列中的元素都能得到利用，则需设置一个标志域tag，并以tag的值为0或1来区分队头指针front和队尾指针rear相同时的队列状态是“空”还是“满”。

试编写与此结构相应的入队和出队算法。

2、算法思想：

1°：在循环队列的类型结构中，增设一个tag的整型变量，进队时置tag为1，出队时置ag为0（因为只有入队操作可能导致队满，也只有出队操作可能导致队空）。

队列Q初始时，置tag=0、front=rear=0。这样队列的4要素如下：

队空条件：Q.front == Q.rear && Q.tag == 0;

队满条件：Q.front == Q.rear && Q.tag == 1;

进队操作：Q.data[Q.rear] = x; Q.rear = (Q.rear+1)%MaxSize; Q.tag = 1;

出队操作：x = Q.data[Q.front]; Q.front = (Q.front+1)%MaxSize; Q.tag = 0;

```
*********************************************************************************************
*/

/*进队操作*/
void EnQueue(SqQueue_1 S1, ElemType x) {
	//判断是否队满
	if (S1.front == S1.rear && S1.tag == 1) {
		printf("The SqQueue_1 is FUll!!!\n");
		return;
	}else {
		S1.data[S1.rear] = x;
		S1.rear = (S1.rear + 1) % MaxSize;
		S1.tag = 1;
	}
```

```
}

/*出队操作*/
void DeQueue(SqQueue_1 S1, ElemType &e) {
        //判断石否队空
        if (S1.front == S1.rear && S1.tag == 0) {
                printf("The SqQueue_1 is NUll!!!\n");
                return;
        }else {
                e = S1.data[S1.front];
                S1.front = (S1.front + 1) % MaxSize;
                S1.tag = 0;
        }
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-3.2 ，应用题第2题（P78）
*********************************
1、题目：
        Q是一个队列，S是一个空栈，实现将队列中的元素逆置的算法；
2、算法思想：
        1°：将队列中的元素依次出队，同时进入栈中
        2°：将栈中的元素依次出栈，同时进入队列，即可完成逆置
*****************************************************************************************
*/

/*逆置这个队列*/
void invertSqQueue(SqStack S, SqQueue &Q) {
        int e;
        while (Q.front != Q.rear) {
                e = Q.data[Q.front];
                Q.front = (Q.front + 1) % MaxSize;
                S.data[++S.top] = e;
        }
        while (S.top != -1) {
                e = S.data[S.top--];
                Q.data[Q.rear] = e;
                Q.rear = (Q.rear + 1) % MaxSize;
        }
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-3.2 ，应用题第3题（P78）
*********************************
```

1、题目：

利用两个栈S1、S2来模拟一个队列，已知栈的4个运算定义如下：

```
Push(S,x);                                  //元素x入栈S
Pop(S,x);                                   //S出栈并将出栈的值赋给x
StackEmpty(S);                              //判断栈是否为空
StackOverflow(S);                           //判      断栈是否满
```

那么如何利用栈的运算来实现该队列的3个运算（形参由读者根据要求自己设计）：

```
Enqueue_1;          //将元素x入队
Dequeue_1;          //出队，并将出队元素存储在x中
QueueEmpty;       //判断队列是台为空
```

2、算法思想：

1°：利用两个栈S1和S2来模拟一个队列，当需要向队列中插入一个元素时，用S1来存放已输入的元素，即S1执行入栈操作。

当需要出队时，则对S2执行出栈操作。由于从栈中取出元素的顺序是原顺序的逆序，所以，必须先将S1中的所有元素全部出栈并入栈到S2中，

再在S2中执行出栈操作，即可实现出队操作，而在执行此操作前必须判断S2是否为空，否则会导致顺序混乱。当栈S1和S2都为空时队列为空。

总结如下：

1）对S2的出栈操作用做出队，若S2为空，则先将S1中的所有元素送入S2；

2）对S1的入栈操作用做入队，若S1满，必须先保证S2为空，才能将S1中的元素全部插入S2中。

```
*********************************************************************************************
*/

/*两个栈来模拟入队
int Enqueue_1(SqStack &S1, SqStack &S2, ElemType e) {
        if (!StackOverflow(S1)) {
```

```
                Push(S1,e);
                return 1;
        }
        if (StackOverflow(S1) && !StackEmpty(S2)) {
                printf("队列满, 无法再入队!!!\n");
                return 0
        }
        if (StackOverflow(S1) && StackEmpty(S2)) {
                int x;
                while (!StackEmpty(S1)) {
                        Pop(S1, x);
                        Push(S2, x);
                }
        }
        Push(S1, e);
        return 1;
}
*/



/*两个栈来模拟出队
void Dequeue_1(SqStack S1, SqStack S2, ElemType &e) {
        if (!StackEmpty(S2)) {
                Pop(S2, e);
        }else if (StackEmpty(S1)) {
                printf("队空,无法再出队!!!\n");
        }else {
                int x;
                while (!StackEmpty(S1)) {
```

```
                Pop(S1, x);
                Push(S2, x);
            }
            Pop(S2, e);
        }
}
*/


/*判断队列是否为空
bool QueueEmpty(SqStack S1, SqStack S2) {
        return StackEmpty(S1) && StackEmpty(S2);
}
*/
```

```
#include "stdafx.h"



//1、创建一个链式队列的节点
LNode* createLNode(ElemType data) {
	LNode* newLNode = (LNode*)malloc(sizeof(LNode));
	if (newLNode) {
		newLNode->data = data;
		newLNode->next = NULL;
	}
	return newLNode;
}


//1-1、创建一个链式队列 (包含着队头指针和队尾指针)
//在初始化里已经整合了


//2-1、初始化一个顺序队列
void InitSqQueue(SqQueue& SQ) {
	SQ.front = SQ.rear = 0;
}

/**
*2-2、初始化一个链式队列(带头结点的)
**/
void InitLinkQueue(LinkQueue &LQ) {
	LQ.front = LQ.rear = (LNode*)malloc(sizeof(LNode));
	if (LQ.front) {
		LQ.front->next = NULL;
```

```
        }

}



//3-1、判断顺序队列是否为空
bool isEmpty(SqQueue SQ) {
        if (SQ.front == SQ.rear) return true;
        else return false;
}

//3-2、判断链式队列是否为空
bool isEmpty(LinkQueue LQ) {
        if (LQ.front == LQ.rear) return true;
        else return false;
}



//4-1、顺序队列 (循环队列) 的进队
void EnQueue(SqQueue &SQ, ElemType x) {
        if ((SQ.rear + 1) % MaxSize == SQ.front) {
                printf("The Queue is full!!!\n");
                return;
        }
        SQ.data[SQ.rear] = x;
        SQ.rear = (SQ.rear + 1) % MaxSize;
}

//4-2、链式队列的进队(无需判断队满)
```

```
void EnQueue(LinkQueue &LQ, ElemType x) {
        LNode *newNode = createLNode(x);
        LQ.rear->next = newNode;
        LQ.rear = newNode;
}


//5-1、顺序队列 (循环队列) 的出队
void DeQueue(SqQueue &SQ, ElemType &e) {
        if (SQ.front == SQ.rear) {
                printf("The Queue is null!!!\n");
                return;
        }
        e = SQ.data[SQ.front];
        SQ.front = (SQ.front +1 ) % MaxSize;
}

//5-2、链式队列的出队 (需判断队空)
void DeQueue(LinkQueue &LQ, ElemType &e) {
        if (LQ.front == LQ.rear) {
                printf("The Queue is null!!!\n");
                return;
        }
        LNode *p = LQ.front->next;
        e = p->data;
        LQ.front->next = p->next;
        if (LQ.rear == p) LQ.rear = LQ.front;
        free(p);
}
```

```
//6-1、读取顺序队列队头元素
ElemType GetHead(SqQueue SQ) {
        if (!isEmpty(SQ)) {
                return SQ.data[SQ.front];
        } else {
                printf("The SqQueue is null in GetHead()!!!\n");
                return NULL;
        }
}

//6-2、读取循环队列的队头元素
ElemType GetHead(LinkQueue LQ) {
        if (!isEmpty(LQ)) {
                return LQ.front->next->data;;
        }else {
                printf("The LinkQueue is null in GetHead()!!!\n");
                return NULL;
        }
}




/******栈系列
**********************************************************************************************
*****************************/



//创建一个链式栈的节点
LSNode* createLSNode(int data) {
```

```
        LSNode* newLSNode = (LSNode*)malloc(sizeof(LSNode));
        if (newLSNode) {
                newLSNode->data = data;
                newLSNode->next = NULL;
        }
        return newLSNode;
}


//创建一个链式栈 (不带头结点)
LinkStack createLinkStack() {
        LinkStack firstLSNode = (LinkStack)malloc(sizeof(LinkStack));
        if (firstLSNode) {
                firstLSNode = NULL;
        }
        return firstLSNode;
}



//初始化一个顺序栈
void InitSqStatck(SqStack& S) {
        S.top = -1;
}

//初始化一个链式栈 (链式栈是不带头结点的)
void InitLinkStatck(LinkStack& LS) {
        LS = NULL;
}
```

```
//判断一个顺序栈是否为空
bool SqStackIsEmpty(SqStack S) {
	return S.top == -1;
}

//判断一个链式栈是否为空
bool LinkStackIsEmpty(LinkStack LS) {
	return LS == NULL;
}



//顺序栈的进栈操作
bool Push(SqStack& S, int x) {
	if (S.top == MaxSize - 1) {
		return false;
	}
	S.data[++S.top] = x;
	return true;
}

//链式栈(不带头结点)的进栈操作
void Push(LinkStack& LS, int x) {
	LSNode* newLSNode = createLSNode(x);
	if (LS) {
		newLSNode->next = LS;
		LS = newLSNode;
	}
	else {
		LS = newLSNode;
	}
```

```
}



//顺序栈的出栈操作
bool Pop(SqStack& S, int& element) {
        if (S.top == -1) {
                return false;
        }
        element = S.data[S.top--];
        return true;
}

//链式栈的出栈操作 (int型)
bool Pop(LinkStack& LS, int& element) {
        if (!LS) {
                return false;
        }
        element = LS->data;
        LS = LS->next;
        return true;
}



//获取顺序栈顶元素
bool Top(SqStack S, int& element) {
        if (S.top == MaxSize - 1) {
                return false;
        }
        element = S.data[S.top];
        return true;
```

```
}

//获取链式栈栈顶元素
bool Top(LinkStack LS, int& element) {
        if (!LS) {
                return false;
        }
        element = LS->data;
        return true;
}



//销毁顺序栈
void ClearSqStack(SqStack& S) {
        S.top = -1;
        //free(*S);
}

//销毁链式栈
void ClearLinkStack(LinkStack& LS) {
        LS = NULL;
        free(LS);
}
```

```
#include "stdafx.h"

/*********************王道2019年chapter-3.3 ，应用题第1题（P87）
*********************************
1、题目：
        假设一个算术表达式中包含圆括号、方括号和花括号3种类型的括号，编写一个算法来判别表达式中的括号是否配对，以字符“\0”作为算术表达式的结束符；
2、算法思想：
        1°：逐个遍历表达式字符序列；
        2°：若当前字符是左括号、左方括号、左花括号，则将其入栈；
        3°：若当前字符是右括号、右方括号、右花括号，则弹出栈顶元素，若与当前右侧同类型括号不匹配，则视为表达式匹配错误，返回false；
*****************************************************************************************
*/

/*匹配括号是否正确*/
bool matchBracket(char *ch) {
        SqStack S;
        S.top = -1;
        int i = 0;
        while (ch[i] != '\0') {
                char e;
                switch (ch[i]){
                        case '(':
                                S.data[++S.top] = '(';
                                break;
                        case '[':
                                S.data[++S.top] = '[';
                                break;
                        case '{':
                                S.data[++S.top] = '{';
```

```
                        break;
                case ')':
                        if (S.data[S.top--] != '(') return false;
                        break;
                case ']':
                        if (S.data[S.top--] != '[') return false;
                        break;
                case '}':
                        if (S.data[S.top--] != '{') return false;
                        break;
                default:
                        break;
            }
            i++;
        }
        if (S.top == -1) {
            return true;
        }else {
            return false;
        }

}
```

```
#include "stdafx.h"

/**********王道2019年chapter-3.3 ，应用题第2题（P87）
**********
```

1、题目：

如图3-18所示铁道进行车厢调度（注意：两侧铁道均为单向行驶道，火车调度站有一个用于调度的“栈道”），

火车调度站的入口处有n节硬座和软座车厢（分别以H和S表示）等待调度，试编写算法，输出对这n节车厢进行调度的操作（即入栈或出栈操作）序列，

以使所有的软座车厢都被调整到硬座车厢之前。

2、算法思想：

1°：两侧的铁道均为单向行驶道，且两侧不相通。所有车辆都必须通过“栈道”进行调度。算法的基本设计思想：

所有车厢依次前进并逐一检查，若为硬座车厢则入栈，等待最后调度。检查完后，所有的硬座车厢已全部入栈道，

车道中的车厢均为软座车厢，此时将栈道的车厢调度出来，调整到软座车厢之后。

```
**********
*/

/*实现火车的调度*/
void trainSchedule(char *train) {
	char* p = train;
	char* q = train;
	char c;
	SqStack S;
	S.top = -1;
	while (*p) {
		if (*p == 'H') {
			S.data[++S.top] = *p;	//把H存入栈中，待将S排好后，再出栈H
		}else {
```

```
                *(q++) = *p;            //把S调到前面
            }
            p++;
        }

        while (S.top != -1){
            c = S.data[S.top--];
            *(q++) = c;                 //把H接在后面
        }
}
```

```
#include "stdafx.h"

/********************王道2019年chapter-3.3 , 应用题第1题 (P88)
*********************************
1、题目:
        利用一个栈实现以下递归函数的非递归计算:
                                = 1,
n=0;
                Pn(x)   =2x,                                                    n=1;
                                =2xP(n-1)(x)-2(n-1)P(n-2)(x)        n>1
2、算法思想:
        1°: 设置一个栈用于保存n和对应的PA (x) 值;
        2°: 栈中相邻元素的Pn (x) 有题中关系;
        3°: 然后边出栈边计算P (x) , 栈空后该值就计算出来了。
*****************************************************************************************
*/

double Pn(int n, double x) {
        struct myStack{
                int no;                 //保存n的值
                double val;                 //保存Pn的值
        }mst[MaxSize];
        int top = -1 , i;
        double fv1 = 1, fv2 = 2 * x;     //n=0, n=1是的初值
        for (i = n; i >= 2; i--) {
                mst[++top].no = i;
        }

        while (top >=0 ) {
                mst[top].val = 2 * x * fv2 - 2 * (mst[top].no - 1) * fv1;
                fv1 = fv2;
```

```
                fv2 = mst[top--].val;
        }

        if (n == 0) return fv1;
        return fv2;
}


double Pn_recursion(int n, double x) {
        if (n == 0) return 1;
        if (n == 1) return 2 * x;
        if (n > 1) return 2 * x * Pn_recursion(n - 1, x) - 2 * (n - 1) * Pn_recursion(n - 2,
x);
}
```

```
#include "stdafx.h"

bool QueueEmpty(SqQueue SQ);
bool QueueFull(SqQueue Q);
void DeQueue(SqQueue Q, int& x);
void EnQueue(SqQueue Q, int x);
/********************王道2019年chapter-3.3 ，应用题第1题 (P88)
*********************************
1、题目:
        某汽车轮渡口，过江渡船每次能载10辆车过江。过江车辆分为客车类和货车类，上渡船有如下规定:
                同类车先到先上船;
//【暗示数据结构用队列】
                客车先于货车上渡船，且每上4辆客车，才允许放上一辆货车；若等待客车不足4辆，则以货车代替;
                若无货车等待，允许客车都上船。试设计一个算法模拟渡口管理。
2、算法思想:
        1°：假设数组Q的最大下标为10，恰好是每次载渡的最大量;
        2°：假设客车的队列为Q1，货车的队列为Q2;
        3°：若Q1充足，则每取4个Q1元素后再取一个Q2元素，直到Q的长度为10;
        4°：    若Q1不充足，则直接用Q2补齐。
*****************************************************************************************
*/

SqQueue q;      //过江渡船载渡队列
SqQueue q1;     //客车队列
SqQueue q2;     //货车队列
/*过江渡船模拟*/
void ferryManagement() {
        int i = 0, j = 0;           // i来计数客车是否满4辆 ,j 表示渡船上的总车辆数
        int x;
```

```
        while (j < 10) {
                if (!QueueEmpty(q1) && i < 4) {   //客车队列不空且未上足4辆车，则继续上客车
                        DeQueue(q1, x);
                        EnQueue(q, x);        //客车上渡船
                        i++;
                        j++;
                }else if (i == 4 && !QueueEmpty(q2)) {   //客车已足4辆且有货车可上，则上货车
                        DeQueue(q2, x);
                        EnQueue(q, x);         //货车商船
                        j++;
                        i = 0;                 //船上客车计数置0
                }else if(j < 10 && QueueEmpty(q2)){              //若车无货车等待，则将所有客车上船
                        while (!QueueEmpty(q1)) {
                                DeQueue(q1, x);
                                EnQueue(q, x);
                                j++;
                        }
                }else {
                        while (j < 10 && i < 4 && !QueueEmpty(q2)) {    //若客车不足4辆，则直接用货车补齐。
                                DeQueue(q2, x);
                                EnQueue(q, x);
                                i++;
                                j++;
                        }
                        i = 0;
                }
        }
        if (QueueEmpty(q1) && QueueEmpty(q2)) {
                j = 11;          //直接退出
```

```
        }
}

bool QueueEmpty(SqQueue Q) {
        return Q.front == Q.rear;
}

bool QueueFull(SqQueue Q) {
        return Q.front == (Q.rear + 1) % MaxSize;
}


/*出队*/
void DeQueue(SqQueue Q, int &x) {
        if (!QueueEmpty(Q)) {
                x = Q.data[Q.front];
                Q.front = (Q.front + 1) % MaxSize;
        }else {
                return;
        }
}

void EnQueue(SqQueue Q, int x) {
        if (!QueueFull(Q)) {
                Q.data[Q.rear] = x;
                Q.rear = (Q.rear + 1) % MaxSize;
        }else {
                return;
        }
}
```

```
#include "stdafx.h"

int calculateAB(int a, char op, int b);

/*********************王道2019年chapter-3.3 , 例题第1题 (p83) ********************************
1、题目:
        给出一个后缀表达式(只能求出多个个位数的混合运算), 求出其数值。
2、算法思想:
        1°: 遍历后缀表达式序列;
        2°: 若当前字符是数字 (0～9),则将其压入栈中;
        3°: 若当前字符是操作符 (+-/*),则从栈中弹出两个数字, 和当前字符进行运算, 运算完后将其压入栈中;
        4°: 所有字符遍历完后, 栈顶元素即为表达式的值
*************************************************************************************************
*/

/*后缀表达式求其具体值*/
int calcPostExpression(char ch[]) {
        SqStack S;
        S.top = -1;
        int i = 0;
        int a, b, result = 0;
        char c;

        while (ch[i] != '\0') {
                if (ch[i] >= '0' && ch[i] <= '9') {
                        S.data[++S.top] = ch[i] - '0';
                }else {
                        b = S.data[S.top--];
                        a = S.data[S.top--];
                        c = ch[i];
```

```
                result = calculateAB(a, c, b);
                S.data[++S.top] = result;
            }
            i++;
        }
        return result;
}
```

```
/*针对操作符求值*/
int calculateAB(int a, char op, int b) {
        switch (op) {
        case '+':
                return a + b;
                break;
        case '-':
                return a - b;
                break;
        case '*':
                return a * b;
                break;
        case '/':
                if (b != 0) {
                        return a / b;
                }
                else {
                        printf("The denominator can't be 0!!!\n");
                        return NULL;
                }
```

```
            break;
        }
        return 0;
}
```

```
#include "stdafx.h"


/*********************王道2019年chapter-3.3 ，例题第2题（p83）*********************************
```

1、题目：

斐波那契数列。

2、算法思想：

递归是一种重要的程序设计方法。简单地说，如果在一个函数、过程或数据结构的定义中又应用了它自身，那么这个函数、过程或数据结构称为是递归定义的，简称递归。

它通常把一个大型的复杂问题，层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的代码就可以描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。但在通常情况下，它的效率并不是太高。

以斐波那契数列为例，其定义为：

$$fib(n)=\begin{cases} fib(n-1)+fib(n-2), & n>1 \\ 1, & n=1 \\ 0, & n=0 \end{cases}$$

```
*********************************************************************************************
*/


int fib(int n) {
	if (n == 0)  return 0;                                                    //
边界条件（递归出口）
	if (n == 1)  return 1;                                                    //
边界条件
	if (n > 1)   return fib(n - 1) + fib(n - 2);          //递归表达式
	return NULL;
}
```

```
#include "stdafx.h"

char Top(MyStatck S);
bool StatckEmpty(MyStatck S);
bool StatckFull(MyStatck S);
void Pop(MyStatck &S, char& x);
void Push(MyStatck& S, char x);

/*********************王道2019年chapter-3.3 , 例题第3题 (p83) ********************************
1、题目:
        中缀表达式转后缀表达式。
2、算法思想:
        1:遍历整个中缀表达式;
        2:若果遇到左侧括号, 则直接入栈;
        3:若遇到右括号则从栈一直弹出元素, 直到弹出与当前右括号匹配的左括号
        4:若遇到 ‘+’ or ‘-’, 则直接出栈;
        5:若遇到 ‘*’ or ‘÷’, 则一直出栈直到‘+’ or ‘-’字符;
        6:若遇到数字, 则
*************************************************************************************
*/



void infixToPost(char infix[]) {
        MyStatck S;
        S.top = -1;

        Post post;
        post.len = 0;

        int i = 0;
```

```
        char x;
        char c;
        while (infix[i] != '\0') {
                c = infix[i++];
                switch (c){
                case '(':
                case '[':
                case '{':
                        Push(S, c);
                        break;
                case ')':
                case ']':
                case '}':
                        while (!StatckEmpty(S) && Top(S) != '(' && Top(S) != '[' && Top(S) !=
'{') {
                                Pop(S, x);
                                post.data[post.len++] = x;
                        }
                        S.top--;
                        break;
                case '+':
                case '-':
                        while (!StatckEmpty(S) && Top(S) != '(' && Top(S) != '[' && Top(S) !=
'{') {
                                Pop(S, x);
                                post.data[post.len++] = x;
                        }
                        Push(S, c);
                        break;
                case '*':
                case '/':
```

```
                        while (!StatckEmpty(S) && Top(S) != '(' && Top(S) != '[' && Top(S) !=
'{' && Top(S) == '/' || Top(S) == '*') {
                                Pop(S, x);
                                post.data[post.len++] = x;
                        }
                        Push(S, c);
                        break;
                default:
                        while (c != NULL) {   //c >= '0' && c <= '9')
                                post.data[post.len++] = c;
                                break;
                        }
                }

        }

        while (!StatckEmpty(S)) {
                Pop(S, x);
                post.data[post.len++] = x;
        }
        //post.data[post.len++] = '\0';

        for (int i = 0; i < post.len; i++) {
                printf("%c", post.data[i]);
        }
}




/*************************************************************************************************
***/
```

```
/*栈判空*/
bool StatckEmpty(MyStatck S) {
        return S.top == -1;
}

/*栈是否满*/
bool StatckFull(MyStatck S) {
        return S.top == MaxSize - 1;
}


/*出栈*/
void Pop(MyStatck &S, char& x) {
        if(!StatckEmpty(S)) x = S.data[S.top--];
}

/*入栈*/
void Push(MyStatck &S, char x) {
        if (!StatckFull(S)) S.data[++S.top] = x;
}


/*获取栈顶元素*/
char Top(MyStatck S) {
        if (!StatckEmpty(S)) return S.data[S.top];
}
```