链表数据结构

```
#ifndef  _Link_H_
#define _Link_H_

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>


//dat
typedef struct ListNode//单向链表节点
{
    int nodeData;//数据
    struct ListNode *nodeNext;//地址
}ListNode,*LinkList;//指向节点的指针
typedef struct ListNodeDDW{//双向链表节点
    struct ListNodeDDW *nodePrev;
    int nodeData;
    struct ListNodeDDW *nodeNext;
}ListNodeDDW,*LinkListDDW;
LinkList ListArg;//链表头节点
ListNode ListOne;//实体节点


//opear
LinkList InitListHead(LinkList L){
    //初始化链表：返回一个指向头节点的链表指针，因为是空节点所以头节点的指针域指向NULL
    ListNode *LN ;int x;
    L = (LinkList)malloc(sizeof(ListNode));
    L->nodeNext = NULL;
    scanf("%d",&x);
    while(x!=-1){
        LN = (ListNode*)malloc(sizeof(ListNode));
        LN->nodeData=x;
        LN->nodeNext=L->nodeNext;
        L->nodeNext=LN;
        scanf("%d",&x);
    }
    return L;
}
LinkList InitListEnd(LinkList L){
    ListNode *LN,*EN; int x;
    L = (LinkList)malloc(sizeof(ListNode));
    L->nodeNext = NULL;
    EN = L;//尾节点
    scanf("%d",&x);
    while(x!=-1){
        LN = (ListNode*)malloc(sizeof(ListNode));
        LN->nodeData = x;
        EN->nodeNext = LN;
        EN = LN;
        scanf("%d",&x);
    }
```

```
    EN->nodeNext = NULL;
    return L;
}
LinkListDDW InitListEndDDW(LinkListDDW  L){//双向循环链表尾插法
    ListNodeDDW *LN,*EN;int x;
    L = (LinkListDDW)malloc(sizeof(ListNodeDDW));
    L->nodeNext = NULL; L->nodePrev = NULL;
    EN = L;//工作指针
    scanf("%d",&x);
    while(x!=-1){
        LN = (ListNodeDDW*)malloc(sizeof(ListNodeDDW));
        LN->nodeData = x;
        LN->nodePrev = NULL;
        LN->nodeNext = NULL;

        EN->nodeNext = LN;
        LN->nodePrev = EN;
        EN = EN->nodeNext;
        scanf("%d",&x);
    }
    EN->nodeNext = L;
    L->nodePrev = EN;

    return L;
}
void ShowListDDW(LinkListDDW L){//通过后续节点遍历
    if(L->nodeNext==L){
        printf("链表为空");
        return;
    }
    LinkListDDW f = L->nodeNext;//指向首元节点
    while(f&&f!=L){
        printf("链表值:%d\n",f->nodeData);
        f = f->nodeNext;
    }
    return ;
}
void ShowList(LinkList L){//循环输出链表值
    if(L->nodeNext==NULL){
        printf("链表为空");
        return;
    }
    LinkList f = L->nodeNext;//指向首元节点
    while(f){
        printf("链表值:%d\n",f->nodeData);
        f = f->nodeNext;
    }
    return ;
}

ListNode getItemUN(LinkList L,int Nx){//根据索引获取节点
    ListNode  *p = L->nodeNext; //指向首元节点
    int f = 1;
    if(Nx == 0){ // 空表返回头节点
        return *L;
    }
    while(p&&f<Nx){ //从首元节点开始循环链表指向下一位
        p=p->nodeNext;
```

```
        f++;
    }
    return *p;
}
ListNode getItemUV(LinkList L,int val){//根据节点值获取节点
    ListNode *p = L->nodeNext;//指向首元节点
    while(p&&p->nodeData!=val){
        p = p->nodeNext;
    }
    return *p;//如果链表为空返回值NULL
}
void ListAddNode(LinkList L,int n,ListNode LN){
    LinkList p = L; int f=0;
    while(p&&f<(n-1)){//表头不为空将链表推至需要插入的前一位
        p=p->nodeNext;
        f++;
    }
    if(!p){//如果插入位置的前一个节点为空则出错
        printf("error");
        return ;
    }
    LN.nodeNext = p->nodeNext;
    p->nodeNext = &LN;
    return ;
}
ListNode ListRmNode(LinkList L,int n){
    LinkList p = L; int f=0;
    while(p&&f<(n-1)){//循环到要删除的前一个元素
        p=p->nodeNext;
        f++;
    }
    if(!p->nodeNext){//如果要删除的元素为空 则返回
        printf("error");
        return *p->nodeNext;
    }
    LinkList delNode = p->nodeNext;
    p->nodeNext = delNode->nodeNext;
    free(delNode);
    return *delNode;
}
LinkList ListRmNodeUseNode(LinkList L){
    LinkList newList;
    newList = L->nodeNext;
    free(L);
    return newList;
}
#endif
```

题目01

将两个递增的有序列表合并为一个递增的有序列表。要求结果链表仍使用原来两个链表的存储空间，不另外占用其它的存储空间。表中不允许有重复的数据

代码

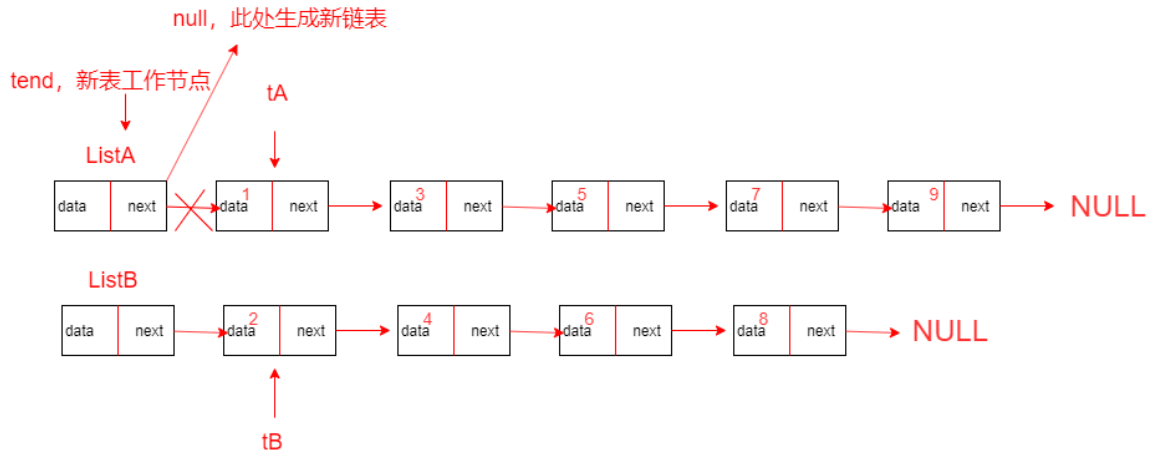
```
void MergeList(LinkList listA,LinkList listB){
    LinkList tA = listA->nodeNext,tB = listB->nodeNext;//指向有数据的节点
    LinkList t,tend; //辅助节点
    listA->nodeNext = NULL ; //为生成新链表指空
    tend = listA; //新表工作节点
    while(tA&&tB){//
        if(tA->nodeData<tB->nodeData){//将比较的小值插入到节点中,后移工作节点
            tend->nodeNext = tA;
            tend = tA;
            tA = tA->nodeNext;
        }else if(tA->nodeData>tB->nodeData){
            tend->nodeNext = tB;
            tend = tB;
            tB = tB->nodeNext;
        }else{
            // 如果出现有相同值得情况下，释放其中一个
            tend->nodeNext = tA;
            tend = tA;
            tA = tA->nodeNext;
            t = tB->nodeNext;//释放
            free(tB);
            tB = t;
        }
    }
    tend->nodeNext = tA?tA:tB;//总有一个节点没有情况加入到链表中
    free(listB);
}
```

运行结果

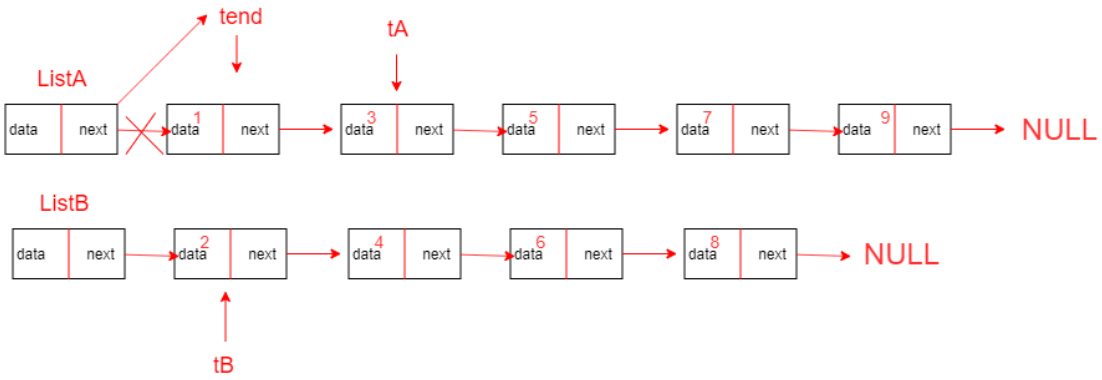
```
int main(){
    LinkList la,lb,lc;
    la = InitListEnd(la);
    lb = InitListEnd(lb);
    MergeList(la,lb);
    ShowList(la);
}
//console
1
3
5
7
9
-1
2
4
6
8
-1
链表值:1
链表值:2
链表值:3
链表值:4
链表值:5
```

```
链表值:6
链表值:7
链表值:8
链表值:9
```

分析



一次运行后的结果，以此类推



算法的时间复杂度O(1) ， 空间复杂度O(1)

题目03

已知两个链表 A 和 B 分别表示两个集合，其元素递增排列。请设计一个算法，用于求出 A 与 B 的交集，并将结果存放在 A 链表中

代码

```
// 已知两个链表 A 和 B 分别表示两个集合，其元素递增排列。
// 请设计一个算法，用于求出 A 与 B 的交集，并将结果存放在 A 链表中
void listAUlistB(LinkList la,LinkList lb){
    LinkList tA = la->nodeNext,tB = lb->nodeNext;
    LinkList tend,t;
    tend = la; // tend???????????????
    while(tA&&tB){
        if(tA->nodeData==tB->nodeData){//??????????????????????
            tend->nodeNext = tA;
            tend = tA;
            tA = tA->nodeNext;
            t = tB;
            tB = tB->nodeNext;
```

```
            free(t);
        }else if(tA->nodeData<tB->nodeData){
            //????????????????????????????????????????????????
            t = tA;
            tA = tA->nodeNext;
            free(t);
        }else{
            t = tB;
            tB = tB->nodeNext;
            free(t);
        }
    }
    while(tA){//????????
        t = tA;
        tA = tA->nodeNext;
        free(t);
    }
    while(tB){
        t = tB;
        tB = tB->nodeNext;
        free(t);
    }
    tend->nodeNext = NULL;
    free(lb);
}
```

运行结果

```
int main(){
    LinkList la,lb,lc;
    la = InitListEnd(la);
    lb = InitListEnd(lb);
    listAUlistB(la,lb);
    ShowList(la);
}
// console
1
2
3
4
-1
2
4
-1
链表值:2
链表值:4
```

分析

实现步骤类似题目01，图省略

算法的时间复杂度O(1)，空间复杂度O(1)

题目05

设计算法将一个带头节点的单链表 A 分解为两个具有相同结构的链表 B 和 C，其中 B 表的节点为 A 表中值小于0的节点，而 C 表的节点为 A 表中值大于0的节点 (链表A中的元素为非零整数，要求B、C表利用A表的节点)

代码

```
// 设计算法将一个带头节点的单链表 A 分解为两个具有相同结构的链表 B 和 C，
// 其中 B 表的节点为 A 表中值小于0的节点，
// 而 C 表的节点为 A 表中值大于0的节点 (链表A中的元素为非零整数，要求B、C表利用A表的节点)
void  ListAtoBandC(LinkList la,LinkList lb,LinkList lc){
    LinkList tA= la->nodeNext,tB = lb,tC = lc; //???????????????
    while(tA){//???????
        if(tA->nodeData == 0){
            return ;//?????0????
        }else if(tA->nodeData<0){//???? B ??
            tB->nodeNext = tA;
            tB = tB->nodeNext;
            tA = tA->nodeNext;
        }else if(tA->nodeData>0){
            tC->nodeNext = tA;
            tC = tC->nodeNext;
            tA = tA->nodeNext;
        }
    }
    tB->nodeNext = NULL;
    tC->nodeNext = NULL;
    free(la);
}
```

运行结果

```
int main(){
    LinkList la,lb,lc;
    la = InitListEnd(la);
    lb = InitListEnd(lb);
    lc = InitListEnd(lc);
    ListAtoBandC(la,lb,lc);
    ShowList(lb);
    ShowList(lc);
}
// console
-2
-4
-6
-8
1
3
5
7
-1
-1
-1
链表值:-2
链表值:-4
链表值:-6
链表值:-8
```

```
链表值:1
链表值:3
链表值:5
链表值:7
```

分析

实现步骤类似题目01，图省略

算法的时间复杂度O(1) ，空间复杂度O(1)

题目07

设计一个算法，将链表中所有节点的链表方向 `原地` 旋转，既要求仅利用原表的存储空间，换句话说，要求算法的空间复杂度为 o(1)

代码

```
// 设计一个算法，将链表中所有节点的链表方向 `原地` 旋转，
// 既要求仅利用原表的存储空间，换句话说，要求算法的空间复杂度为 o(1)
void ListReverse(LinkList la){
    LinkList tA = la->nodeNext,trear,tpre;//?????????????
    tpre = NULL ; //????? ???????????????
    la->nodeNext = NULL;
    while(tA){
        trear = tA->nodeNext;//?????????
        tA->nodeNext = tpre;
        tpre = tA;
        tA = trear;
    }
    la->nodeNext = tpre;
}
```

运行结果

```
int main(){
    LinkList la,lb,lc;
    la = InitListEnd(la);
    ListReverse(la);
    ShowList(la);
}
// console
1
3
5
7
-1
链表值:7
链表值:5
链表值:3
链表值:1
```

分析

实现步骤类似题目01，图省略

算法的时间复杂度O(1)，空间复杂度O(1)

题目09

已知 p 指向双向循环链表中的一个节点，其节点结构为 `data`、`prior`、`next` 三个域，设计算法 `change(p)`，交换 p 所指向的节点及其前驱节点的顺序

代码

```
// 已知 p 指向双向循环链表中的一个节点，其节点结构为 `data`、`prior`、`next` 三个域，
// 设计算法 `change(p)` ，交换 p 所指向的节点及其前驱节点的顺序
void change(LinkListDDW Lnode){ //因为是循环链表所有对链表和链尾节点不考虑？
    LinkListDDW Pnode,PPnode,Nnode;
    Pnode = Lnode->nodePrev;
    PPnode = Pnode->nodePrev;
    Nnode = Lnode->nodeNext;
    PPnode->nodeNext = Lnode;
    Lnode->nodePrev = PPnode;
    Lnode->nodeNext = Pnode;
    Pnode->nodePrev = Lnode;
    Pnode->nodeNext = Nnode;
    Nnode->nodePrev = Pnode;
}
```

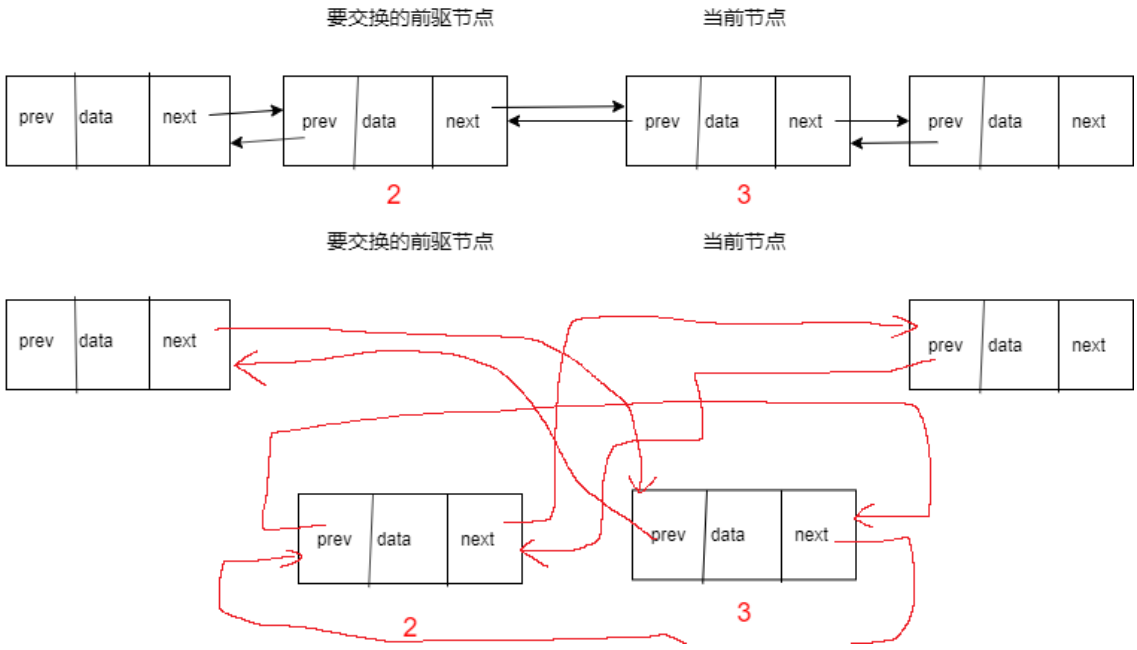
运行结果

```
int main(){
    LinkListDDW la,lb;
    la=InitListEndDDW(la);
    ShowListDDW(la);
    lb = la->nodeNext->nodeNext->nodeNext->nodeNext;//模拟取第4个节点
    change(lb);//调换与其前驱节点的位置
    printf("调换后\n");
    ShowListDDW(la);
}
// console
1
2
3
4
-1
链表值:1
链表值:2
链表值:3
链表值:4
调换后
链表值:1
链表值:2
链表值:4
链表值:3
```

分析

对链表的指针域指向改变来交换位置

第一步



算法的时间复杂度O(1)，空间复杂度O(1)

02

将两个非递减的有序链表合并为一个非递增的有序列表。要求结果链表仍使用原来两个存储空间，不另外占用其它的存储空间。表中允许有重复数据

04

已知两个链表 A 和 B 分别表示两个集合，其元素递增排列。请设计两个算法求出两个集合 A 和 B的差集(仅由在 A 中出现而不在 B中出现的元素所构成的集合)，并将结果以同样的形式存储，同时返回该集合的元素个数

06

设计一个算法，通过一趟遍历确定长度为 n 的单链表中值最大的节点

08

设计一个算法，删除递增有序链表中值大于 `mink` 且小于 `maxk` 的所有元素 (`mink` 和 `maxk` 是给定的两个参数，其值可以和表中的元素相同，也可以不同)

10

已知长度为 n 的线性表 A 采用顺序存储结构，请设计一个时间复杂度为O(n)、空间复杂度为 O（1）的算法，该算法可删除线性表中所有值为 `item` 的数据元素