##### 以下函数在下面各题中均有出现，一次性做出说明，后面不再注解

**1.定义结构体**

**typedef struct ListNode{**

**int val;**

**struct ListNode \*next;**

**}ListNode;**

**2.创建链表函数（输入-1时表示输入结束）**

**ListNode\* creat()**

**{**

**ListNode \* p1,\* p2,\* head;**

**p1 = p2 = (ListNode \* )malloc(LEN);// 开辟新单元**

**cin >> p1 -> val;**

**head = NULL;//将头结点定义为空**

**while(p1 -> val !=-1)**

**{**

**if(head == NULL)head = p1;//head定义为第一个结点**

**else p2 -> next = p1;//链表连接起来**

**p2 = p1; //p2存储p1**

**p1 = (ListNode \* )malloc(LEN);//开辟动态单元**

**cin >> p1 -> val;**

**}**

**p2 ->next = NULL;//最后一个节点为空**

**return head;//返回这个原链表的头指针**

**}**

**3.输出函数**

**void print(ListNode\* h)**

**{**

**if(h == NULL){puts("NULL");return;}**

**while(h){**

**cout << h -> val << ' ';**

**h = h -> next;**

**}**

**puts("");**

**}**

##### 第一题

//反转一个单链表

#include <iostream>

#include <cstring>

#include <cstdlib>

using namespace std;

#define LEN sizeof(struct ListNode)

//定义结构体

typedef struct ListNode{

    int val;

    struct ListNode \*next;

}ListNode;

//read函数为输入链表函数

ListNode\* creat()

{

    ListNode \* p1,\* p2,\* head;

    p1 = p2 = (ListNode \* )malloc(LEN);// 开辟新单元

    cin  >> p1 -> val;

    head = NULL;//将头结点定义为空

    while(p1 -> val !=-1)

    {

        if(head == NULL)head = p1;//head定义为第一个结点

        else p2 -> next = p1;//链表连接起来

        p2 = p1;            //p2存储p1

        p1 = (ListNode \* )malloc(LEN);//开辟动态单元

        cin >> p1 -> val;

    }

    p2 ->next = NULL;//最后一个节点为空

    return head;//返回这个原链表的头指针

}

ListNode\* reverseList(ListNode\* head)

{

    ListNode \* current, \* prev,\*  temp;

    //current已经反转到的节点的位置

    //prev存储current的前一个结点

    //temp存储current的下一个结点

    current = head;

    prev = NULL;

    while( current )//当进行到最后一个空结点时返回

    {

        temp = current -> next;// temp存储current的下一个结点

        current -> next = prev;//将current即当前结点的next指针指向其前一个元素,断开原来的链表关系

        prev = current;//再将prev往后移1为

        current = temp;//对应current也要沿原链表的方向往后移动一位

    }

    return prev;//此时的prev就是反转后的链表的头指针

    //而head指针时原链表的头指针，即新链表的最后一个指针

}

int main()

{

    ListNode  \* head, \*new\_head, \* p;

    head = creat();//head为输入链表的头节点

    new\_head = reverseList(head);//new\_head为反转后的新链表的头节点

    p = new\_head;

    //按序输出反转后的链表

    while(p !=NULL){cout << p -> val << ' '; p = p -> next;}

    return 0;

}

**输入格式（输入一系列数字，输入-1时表示结束）**

**如输入**

**1 2 3 4 5 -1**

**输出**

##### 第二题

//给定一个排序链表，删除所有重复的元素，使得每个元素只出现一次。

//由于此题所给的链表时已经排好序的，所以查找相同元素只需要在相邻结点内查找

#include<iostream>

#include<cstdlib>

using namespace std;

#define  LEN  sizeof(struct ListNode)

//定义结构体数组

typedef struct ListNode{

    int val;

    struct ListNode \*next;

}ListNode;

//creat函数用来创建动态链表

ListNode\* creat()

{

    ListNode \* p1,\* p2,\* head;

    p1 = p2 = (struct ListNode \* )malloc(LEN);//开辟新单元

    cin >> p1 -> val;

    head = NULL;

    while(p1 -> val != -1)//当输入0时表示程序输入结束

    {

        if( head == NULL )head = NULL;

        else p2 -> next = p1;//不是第一次输入时就将原来p1的指针(现在是p2)指向p1,使链表连接起来

        p2 = p1;//将p2存储为原来的p1指针

        p1 = (struct ListNode \* )malloc(LEN);//创建动态链表

        cin >> p1 -> val;

    }

    p2 -> next = NULL;//最后的指针指向的应该为空

    return head;//返回头指针

}

//删除操作只需查找每个结点上的数是否与它的上一个结点的数相等

ListNode\* deleteDuplicates(ListNode\* head)

{

    ListNode  \* current ,\* prev,\* temp;

    //currnet表示现在已经查询到了的结点的位置

    current = head -> next;//开始时应该是第二个结点的位置

    prev = head;//prev存储current的上一个结点，开始时应该是头结点

    while(current)//当查询到最后一个空结点时，删除操作完成

    {

        temp = current -> next;//temp存储current的下一个结点

        if(current -> val == prev -> val)prev -> next = temp;

        //如果相同,则将prev的next指针指向temp，即跳过了current,且此时prev指针应该不变

        else prev = current;//如果不同，prev可向后移动一位

        current = temp;//当前查询结束，current向后移动一位

    }

    return head;//返回头指针

}

int main()

{

    ListNode \* head,\* new\_head,\* p;

    head = creat();//head为创建的原链表的头指针

    p = new\_head = deleteDuplicates(head);//new\_head为删除相同元素后的新链表的头指针

    //输出新链表的元素

    while(p){

        cout << p -> val << ' ';

        p = p -> next;

    }

    return 0;

}

**输入格式(输入一系列的整数，当输入0时表示输入结束)**

**如输入**

**1 1 1 2 2 5 5 5 -1**

**输出1 2 5**

##### 第三题

//3.给定一个排序链表，删除所有含有重复数字的节点，只保留原始链表中 没有重复出现的数字

//此题只需在第二题的基础上修改删除函数即可

#include<iostream>

#include<cstdlib>

using namespace std;

#define  LEN  sizeof(struct ListNode)

//定义结构体数组

typedef struct ListNode{

    int val;

    struct ListNode \*next;

}ListNode;

//creat函数用来创建动态链表

ListNode\* creat()

{

    ListNode \* p1,\* p2,\* head;

    p1 = p2 = (ListNode \* )malloc(LEN);//开辟新单元

    cin >> p1 -> val;

    head = NULL;

    while(p1 -> val != -1)//当输入0时表示程序输入结束

    {

        if(head == NULL)head = p1;

        else p2 -> next = p1;//不是第一次输入时就将原来p1的指针(现在是p2)指向p1,使链表连接起来

        p2 = p1;//将p2存储为原来的p1指针

        p1 = (ListNode \* )malloc(LEN);//创建动态链表

        cin >> p1 -> val;

    }

    p2 -> next = NULL;//最后的指针指向的应该为空

    return head;//返回头指针

}

void print(ListNode\* h)

{

    if(h == NULL){puts("NULL");return;}

    while(h){

        cout << h -> val << ' ';

        h = h -> next;

    }

    puts("");

}

//删除操作只需查找每个结点上的数是否与它的上一个结点的数相等

ListNode\* deleteDuplicates(ListNode\* head)

{

    ListNode  \* current,\* prev,\* temp,\* p;

    current = head -> next;//current初始为head的next指针

    //先找出head的位置，即找到第一个只出现一次的数字作为头结点

    //当head的val==current的val，则head直接定为current -> next

    bool flag = false;//flag用来判断原链表中的head->val是否只出现过一次

    while(head && current && head -> val == current -> val)

    {

        flag = true;

        temp = current -> next;//temp存储current的下一个指针

        head = current;

        current = head -> next;//current初始为head的下一个指针

    }

    //如果flag==true 则此时的head应该也被删除，此时的current才是第一个只出现一次的数

    if(flag)head = current;

    current = NULL;//将current初始为空，防止链表中只有一个已经出现的数字

    if(head)prev = head -> next;//head不为空，prev初始为head的next指针

    if(prev)current = prev -> next;//prev不为空，则current初始为prev的下一个指针

    p = head;//p用来存储当前 距离 查询到的结点 最近的 只出现一次的结点

    //这样prev的前一个结点存的数一定是只出现过一次的数

    while(current)//当查询到最后一个空结点时，删除操作完成

    {

        temp = current -> next;//temp每次都存储current的下一个结点

        if(current -> val == prev -> val) //如果当前结点的val==前一个结点

        {

            p -> next = temp;//删除这两个结点

            p = temp;//p向后移

            current = NULL;//current初始为空

            if(p)prev = p -> next;

            if(prev)current = prev -> next;

        }

        else //如果当前结点不等于前一个结点，则prev->val一定是只出现过一次的数字

        {

                p = prev; //p向后移动

                prev = current;//prev也向后移动

                current = temp;//当前查询结束，current向后移动一位

        }

    }

    return head;//返回头指针

}

int main()

{

    ListNode \* head,\* new\_head,\* p;

    head = creat();//head为创建的原链表的头指针

    new\_head = deleteDuplicates(head);//new\_head为删除相同元素后的新链表的头指针

    print(new\_head);

    return 0;

}

##### 第四题

**/\*测试数据**

**5 5 5 -1**

**5 5 5 -1**

**\*/**

**//代码如下：**

/\*.给出两个非空的链表用来表示两个非负的整数。

其中，它们各自的位数是按照 逆序 的方式存储的，并且它们的每个节点只能存储 一位数字。

如果我们将这两个数相加起来，则会返回一个新的链表来表示它们的和。

您可以假设除了数字0之外，这两个数都不会以0 开头。

\*/

#include <iostream>

#include<cstdlib>

#define LEN sizeof(struct ListNode)

using namespace std;

int n,m;

typedef struct ListNode{

    int val;

    struct ListNode \* next;

}ListNode;

//creat函数为创建新链表的函数

ListNode\* creat()

{

    ListNode \* p1,\* p2,\* head;

    p1 = p2 = (ListNode \*)malloc(LEN);//开辟新单元

    cin >> p1 -> val;

    head = NULL;

    while(p1 -> val != -1)

    {

        if(head == NULL)head = p1;

        else p2 -> next = p1;

        p2 = p1;

        p1 = (ListNode \*)malloc(LEN);//创建动态单元

        cin >> p1 -> val;

    }

    p2 ->next = NULL;//结束链表

    return head;//返回头结点

}

//输出函数

void print(ListNode \*h)

{

    while(h){

        cout << h -> val << ' ';

        h = h -> next;

    }

    puts("");

}

ListNode\* add\_two\_numbers(ListNode\* h1,ListNode\* h2)

{

    ListNode \* head,\* p1,\* p2,\* p,\* pp;

    p1 = h1, p2 = h2;

    head = NULL;

    int sum;//两数之和

    int flag = 0;//sum对10求余的余数，只有0 1两种取法，初始为0

    while(p1 || p2)

    {

        p = (ListNode \*)malloc(LEN);//开辟动态数组

        sum = flag;//如果上一位大于10，则sum初始为1，否则为0

        //每次取两个链表上相同节点的一个数，再将两个指针向后移动1位

        if(p1){

            sum += p1 -> val;

            p1 = p1 -> next;

        }

        if(p2){

            sum += p2 -> val;

            p2 = p2 -> next;

        }

        p ->val = sum % 10;//将sum的各位赋给新链表的指针内

        if(head == NULL)head = p;

        else pp -> next = p;//将上一个指针与之链接起来

        flag = sum / 10;//flag取sum的十位上的数

        pp = p;//pp用来存储p指针

        //如果最后一个sum大于10,则需要在最后再开辟一个单元，存储val = 1；

        if(!p1 && !p2 && flag)

        {

            p = (ListNode \*)malloc(LEN);

            p -> val = 1;

            pp ->next  = p;

            pp = p;

        }

    }

    p -> next = NULL;//使链表完整

    return head;//返回头节点

}

int main()

{

    ListNode \* head1,\* head2,\* head;

    head1 = creat();head2 = creat();//创建两个新链表

    head = add\_two\_numbers(head1,head2);

    print(head);

    return 0;

}

**测试数据1**

**5 5 5 -1**

**5 5 5 -1**

**输出（555 + 555 = 1110）**



**测试数据2（55555 + 4555 = 60110）**

**5 5 5 5 5 -1**

**5 5 5 4 -1**

**输出**



##### 第五题

#include<iostream>

#include<cstdlib>

#define LEN sizeof(struct ListNode)

using namespace std;

typedef struct ListNode{

    int val;

    struct ListNode\* next;

}ListNode;

ListNode\* creat()//创建动态链表

{

    ListNode \* head,\* p1,\* p2;

    p1 = p2 = (ListNode \*)malloc(LEN);//开辟新单元

    head = NULL;

    cin >> p1 -> val;

    while(p1 -> val != -1)//当输入-1时表示程序结束

    {

        if(head == NULL)head = p1;

        else p2 -> next = p1;

        p2 = p1;

        p1 = (ListNode \*)malloc(LEN);//创建动态链表

        cin >> p1 -> val;

    }

    p2 -> next = NULL;

    return head;

}

//print输出链表

ListNode\* print(ListNode \* h)

{

    while(h){

        cout << h -> val <<' ';

        h = h-> next;

    }

    puts("");

}

ListNode\* oddEvenList(ListNode\* head)

{

    ListNode \* p , \* h , \* p1, \* p2;

    p = head;

    h = NULL;

    p1 = p2 = (ListNode \*)malloc(LEN);

    for(int i = 1; p ;i++)

    {

        p1 = (ListNode \*)malloc(LEN);

        //当前结点编号位奇数时，按序存进新链表

        if(i%2 == 1)

        {

            p1 -> val = p -> val;

            if(h == NULL)h = p1;

            else p2 -> next = p1;

            p2 = p1;

        }

        p = p -> next;

    }

    p = head;//p重置为头指针

    for(int i = 1; p ;i++)

    {

        p1 = (ListNode \*)malloc(LEN);

        //当前指针编号位偶数时，按序存储进新链表

        if(i%2 == 0)

        {

            p1 -> val = p -> val;

            p2 -> next = p1;//由于此时为上面的最后的奇数结点，故直接连接

            p2 = p1;

        }

        p = p -> next;

    }

    p2 -> next =NULL;//完成收尾

    return h;//返回头结点

}

int main()

{

    ListNode \* head1,\* head;

    head1 = creat();//创建新链表

    head = oddEvenList(head1);//反转

    print(head);//输出

    return 0;

}

**测试数据1（输入一系列的整数，输入-1时表示输入结束）**

**1 2 3 4 5 -1**

**输出**

**测试数据2**

**1 2 3 4 5 6 -1**

**输出**

**思路：先将原链表中的奇数结点传入新链表，再将偶数结点传入即可**

##### 第六题

/\*

编写程序以x为基准分割链表，使得所有小于x的节点排在大于或等于x的节点之前。

如果链表中包含x，x只需出现在小于x的元素之后。

分割元素x只需处于“右半部分”即可，其不需要被置于左右两部分之间。

\*/

#include <iostream>

#include <cstdlib>

#define LEN sizeof(struct ListNode)

using namespace std;

typedef struct ListNode{

    int val;

    struct ListNode \* next;

}ListNode;

ListNode \* creat()//创建链表

{

    ListNode \*head, \* p1, \* p2;

    p1 = p2 =(ListNode \*)malloc(LEN);

    head = NULL;

    cin >> p1 -> val;

    while(p1 -> val != -1)

    {

        if(head == NULL)head = p1;

        else p2 -> next = p1;

        p2 = p1;

        p1 = (ListNode \*)malloc(LEN);

        cin >> p1 -> val;

    }

    p2 -> next = NULL;

    return head;

}

void print(ListNode \* h)//输出链表

{

    if(h == NULL){puts("NULL");return;}

    while(h){

        cout << h -> val << ' ';

        h = h -> next;

    }

    puts("");

}

ListNode \* partition(ListNode \* h , int x )

{

    ListNode \* p, \* p1, \* p2, \* head;

    p = h;

    head = NULL;

    p1 =(ListNode \*)malloc(LEN);

    p2 = NULL;//防止链表中没有比x小的数

    //先找小于x的结点

    while(p)

    {

        if(p -> val < x)

        {

            p1 -> val = p -> val;//赋值给新链表

            if(head == NULL)head = p1;

            else p2 -> next = p1;

            p2 = p1;

            p1 = (ListNode \*)malloc(LEN);

        }

        p = p -> next;//每次将指针后移一位

    }

    p = h;//重置为头结点

    //再找大于x的结点

    while(p)

    {

        if(p -> val >= x)

        {

            p1 -> val = p -> val;

            if(p2 == NULL)head = p1;

            else p2 -> next = p1;

            p2 = p1;

            p1 = (ListNode \*)malloc(LEN);

        }

        p = p -> next;

    }

    p2 -> next = NULL;//是链表完整

    return head;

}

int main()

{

    int x;

    cin >> x;

    ListNode \* h, \* head;

    h = creat();//创建一个动态链表

    head = partition(h,x);

    print(head);

    return 0;

}

**测试数据1（第一行输入数据x，第二行输入链表各节点内的数，输入-1表示结束输出）**

**5**

**3 5 8 5 10 2 1 -1**

**输出**

**测试数据2**

**5**

**5 6 7 10 9 -1**

**输出**

**思路：创建链表后，依次从表头的位置往后搜索，若当前位置<x，则将其数据传入新链表中，在从表头位置从新搜索找>x的数并传入新链表中，注意：可能会出现原链表中没有比x小的数，故要多增加判断条件**

##### 第七题

/\*

7.给你一个链表的头节点 head.

反复删去链表中由总和值为0的连续节点组成的序列，直到不存在这样的序列为止。

删除完毕后，请你返回最终结果链表的头节点。

你可以返回任何满足题目要求的答案。

\*/

//思路：依次搜索链表，当搜索到一个结点时，求该节点的所有后缀和

//        若有后缀和为0的，则直接删去这段后缀

#include <iostream>

#include <cstdlib>

#include <cstring>

#define LEN sizeof(struct ListNode)

using namespace std;

const int N = 500;

int g[N][N];//存储从当前节点到头结点之间的一段链表中所有后缀和

typedef struct ListNode{

    int val;

    struct ListNode \* next;

}ListNode;

ListNode \* creat()

{

    ListNode \* head, \* p1, \* p2;

    p1 = p2 = (ListNode \*)malloc(LEN);

    head = NULL;

    cin >> p1 -> val;

    while(p1 -> val != 9999)

    {

        if(head == NULL)head = p1;

        else p2 -> next = p1;

        p2 = p1;

        p1 = (ListNode \*)malloc(LEN);

        cin >> p1 -> val;

    }

    p2 -> next = NULL;

    return head;

}

void print(ListNode \*h)

{

    if(h == NULL){puts("NULL");return;}

    while(h){

        cout << h -> val << ' ';

        h = h -> next;

    }

    puts("");

}

ListNode \* delete\_the\_0(ListNode \* head)

{

    if(head == NULL)return NULL;//如果最后没有元素剩下，则返回NULL;

    ListNode \* p = head;

    for(int i = 0 ; p ; i ++)//p为当前搜索到的节点

    {

        g[i][0] = p -> val;//长度为0时，后缀就是其本身

        //j表示后缀长度

        for(int j = 1 ; j <= i ;j ++)g[i][j] = g[i-1][j-1] + p -> val;

        //i表示节点编号，当后缀长大于0，将它前面一个元素的所有后缀+当前数字来得到

        for(int j = 0 ; j <=i ; j ++)

        {

            //搜索当前节点的所有后缀和中是否有等于0的

            if(g[i][j] == 0)

            {

                //当后缀长度等于节点编号时，表示此后缀包含了头节点，头结点也需要删去

                if(i == j) return delete\_the\_0(p -> next);//直接删去p前面一段

                //当后缀长度小于节点编号时，则只需删除这段后缀即可

                ListNode \* temp = head ;

                for(int t = 1 ; t < i -j  ; t ++)temp = temp -> next;//找到后缀起点的前一个结点

                temp -> next = p -> next;//删除

                return delete\_the\_0(head);//下一次重新搜索

            }

        }

        p = p -> next;

    }

    return head;

}

int main()

{

    ListNode \* head , \* newhead;

    head = creat();//创建链表

    newhead = delete\_the\_0(head);

    print(newhead);//输出

    return 0;

}

**测试数据1（输入一个链表，输入9999时表示输入结束，链表长度<500）**

**1 -1 2 -2 3 -3 9999**

**输出NULL**

**测试数据2**

**1 2 3 -3 -2 4 9999**

**输出1 4**

**思路主要思路见注解，现解释后缀含义：如1->2->3->4->NULL，1的后缀只有1**

**2的后缀有1->2和2，3的后缀有3, 2->3 , 1->2->3，后缀和就是后缀里所有数字的和**

##### 第八题

//将两个升序链表合并为一个新的升序链表并返回。

//新链表是通过拼接给定的两个链表的所有节点组成的。

#include <iostream>

#include <cstdlib>

#define LEN sizeof(struct ListNode)

using namespace std;

typedef struct ListNode{

    int val;

    struct ListNode \* next;

}ListNode;

ListNode \* creat()

{

    ListNode \* head, \* p1, \* p2;

    p1 = p2 = (ListNode \*)malloc(LEN);

    head = NULL;

    cin >> p1 -> val;

    while(p1 -> val != -1)

    {

        if(head == NULL)head = p1;

        else p2 -> next = p1;

        p2 = p1;

        p1 = (ListNode \*)malloc(LEN);

        cin >> p1 -> val;

    }

    p2 -> next = NULL;

    return head;

}

void print(ListNode \*h)

{

    if(h == NULL){puts("NULL");return;}

    while(h){

        cout << h -> val << ' ';

        h = h -> next;

    }

    puts("");

}

ListNode \* mergeTwoLists(ListNode \* h1 , ListNode \* h2)

{

    ListNode \* p , \* pp , \* current;

    //默认h1为最终链表的头指针，即h1的数据一定要不大于h2的数据

    //如果开始时h1的数据大，则将两个链表的头指针互换

    //这样新链表的表头就可以确定为h1

    if(h2 -> val < h1 -> val)

    {

        ListNode \* temp = h1;

        h1 = h2;

        h2 = temp;

    }

    //由于h1的位置已经固定，故从h1所在链表开始搜索

    //currnt为当前

    //

    p = h1 ;//p为h1所在链表的当前查询结点current的前一个指针

    pp = h2 ;//pp为h2所在链表的已经搜索到的节点

    current = p -> next;//current为h1所在链表的已经搜索到的节点

    while(p && pp)

    {

        if(current && current -> val <= pp -> val)

        {

            p = current;

            current = current -> next;

        }//按顺序后移

        else

        {

            //在current和p之间加入pp指针

            p -> next = pp;

            p = pp;

            pp = pp -> next;//pp后移一位

            p -> next = current;

        }

    }

    return h1;

}

int main()

{

    ListNode \* head1, \* head2, \* head;

    head1 = creat(); head2 = creat();//创建两个新链表

    head = mergeTwoLists(head1 , head2);

    print(head);

    return 0;

}

**测试数据（输入两个链表，每个链表以-1表示结束输入）**

**2 4 6 8 10 -1**

**1 3 5 7 9 -1**

**输出**

##### 第九题

//判断一个链表是否为回文链表

//思路：回文链表与将它反转之后的链表完成相同

//故可以线创建一个和原链表一样的链表，再将原链表反转

//最后比较两个链表是否对应节点内的数相同，只要有一个不同，就说明不是回文链表

#include <iostream>

#include <cstdlib>

#define LEN sizeof(struct ListNode)

using namespace std;

typedef struct ListNode{

    int val;

    struct ListNode \* next;

}ListNode;

ListNode \* creat()

{

    ListNode \* head, \* p1, \* p2;

    p1 = p2 = (ListNode \*)malloc(LEN);

    head = NULL;

    cin >> p1 -> val;

    while(p1 -> val != -1)

    {

        if(head == NULL)head = p1;

        else p2 -> next = p1;

        p2 = p1;

        p1 = (ListNode \*)malloc(LEN);

        cin >> p1 -> val;

    }

    p2 -> next = NULL;

    return head;

}

void print(ListNode \*h)

{

    if(h == NULL){puts("NULL");return;}

    while(h){

        cout << h -> val << ' ';

        h = h -> next;

    }

    puts("");

}

bool isPalindrome(ListNode \*head)

{

    //先创建一个和原链表一模一样的链表

    ListNode \* newhead , \* p1 , \* p2 , \* p;

    p = head , newhead = NULL;

    p1 = p2 =(ListNode \*)malloc(LEN);

    while(p)

    {

        p1 -> val = p -> val;

        if(newhead == NULL)newhead = p1;

        else p2 -> next = p1;

        p2 = p1;

        p1 = (ListNode \*)malloc(LEN);

        p = p -> next;

    }

    p2 -> next = NULL;

    //该链表的表头定义为newhead

    //下面将原链表反转，反转原理可见题一

    ListNode  \* prev , \* current , \* temp;

    prev = head;

    current = prev -> next;

    while(current)

    {

        temp = current -> next;

        current -> next = prev;

        prev = current;

        current = temp;

    }

    head -> next = NULL;

    head = prev;

    //反转完成后，将head定义为反转后链表的表头

    //最后再比较两个链表是否完全相同

    ListNode \* t1 , \* t2;

    t1 = head ,t2 = newhead;

    while(t1 && t2)

    {

        if(t1 -> val == t2 -> val)//对应位置如果相同，则查询下一位

        {

            t1 = t1 -> next;

            t2 = t2 -> next;

        }

        else return false;//如果对应位置不同，则直接返回false

    }

    return true;//若全都相同，返回true

}

int main()

{

    ListNode \* head , \* h;

    head = creat();

    if(isPalindrome(head))puts("true");

    else puts("false");

    return 0;

}

**测试数据1（输入一个链表，输入-1表示输入结束）**

**1 2 3 3 2 1 -1**

**输出 true**

**测试数据2**

**1 2 3 3 3 -1**

**输出false**

**思路：将原链表与它反转之后的链表进行比较，回文链表反转之后与原链表时一模一样的，反转过程与题1一样！**