

链表

代码题 1

题干

- 题目：

单链表默认结构体：

```
typedef struct LNode
{
    int data;
    struct LNode *next;
} LNode, *Linklist;
```

1、设计一个递归算法，删除不带头节点的单链表 L 中所有值为 x 的结点

- 总结：

- 设计递归算法按照以下两步：
 - (1) 定义终止条件；
 - 当前链表为空时，无需再往下执行，函数返回；
 - (2) 写执行逻辑；
 - 如果当前节点的值为 x 时，执行删除，将节点指向下一个节点；
 - 如果当前节点的值不为 x 时，直接传入当前的 `L->next` 指针，往后进行判断；

解答

```
void Delete(LNode* &l,int x)
{
    if(l==null)
    {
        return;
    }
    if(l->data==x)
    {
        LNode* q = l;
        l = l->next;
        free(q);
        Delete(l,x);
    }
    else
    {
        Delete(l->next,x);
    }
}
```

```
}  
}
```

代码题 2

题干

- 题目：
 - 2、删除带头节点单链表中所有值为 x 的结点
- 总结：
 - 用 p 指针对链表进行遍历，判断 p 指针指向的下一个结点的值是否等于 x，如果等于，就进行删除操作，让 p 指向 `p->next` 指向的下一个结点；

解答

```
void Delete(LNode* &l,int x)  
{  
    LNode* p = l;  
  
    while(p->next!=null)  
    {  
        if(p->next->data==x)  
        {  
            LNode* q = p->next;  
            p->next = q->next;  
            free(q);  
        }  
        else  
        {  
            p = p->next;  
        }  
    }  
}
```

代码题 3

题干

- 题目：
 - 3、删除带头节点单链表中第一个值为 x 的结点
- 总结：
 - 遍历单链表，值为 x 时、执行删除并返回 1。如果遍历完成都没找到，返回 0；

解答

```

int DeleteNumber(LNode* &l,int x)
{
    LNode* p = l;

    while(p->next!=null)
    {
        if(p->next->data == x)
        {
            LNode* q = p->next;
            p->next = q->next;
            free(q);
            return 1;
        }
        else
        {
            // 别漏了指针往后走
            p=p->next;
        }
    }

    // 遍历完后没有找到
    return 0;
}

```

代码题 4

题干

- 题目：
 - 4、从尾到头反向输出单链表每个结点的值
- 总结：
 - 使用递归，先递归的访问到最后一个结点，每一层返回的时候返回结点的值；

解答

```

void PrintInvertNumber(LNode* l)
{
    if(l->next!=null)
    {
        PrintInvertNumber(l->next);
    }

    std::cout<<l->data<<" ";
    return;
}

```

代码题 5

题干

- 题目:

5、试编写算法将单链表就地逆置

- 带头节点;
- 总结:
 - 使用头插法;
 - 遍历链表, 将每一个元素插到当前的头指针的前面;
 - **q** 指针用于暂存当前 **p** 结点的下一个结点;

解答

```
void Invert(LNode* &l)
{
    // 当前p结点从头节点后的第一个结点开始
    LNode* p = l->next;
    l->next = null;

    while(p!=null)
    {
        // q指针用于存储当前p结点的下一个结点信息, 因为接下来p结点会用于将当前结点头插
        LNode* q=p;
        q=q->next;

        // 进行头插
        p->next = l->next;
        l->next = p;

        // p指向下一个结点
        p = q;
    }
}
```

代码题 6

题干

- 题目:

6、从链表中删除给定值在 s 到 t 之间 (不包含 s 和 t) 的所有元素
- 总结:
 - 和代码题 2 思想一致, 只是将删除的条件进行修改;

解答

```
void DeleteRange(LNode* &l,int min,int max)
{
    LNode* p = l;
```

```

while(p->next!=null)
{
    if(p->next->data>min&&minP->next->data<max)
    {
        LNode* q = p->next;
        p->next = q->next;
        free(q);
    }
    else
    {
        p = p->next;
    }
}
}

```

代码题 7

题干

- 题目：

7、试编写在带头结点的单链表 L 中删除最小值点的高效算法（已知最小值唯一）

- 总结：

- 设置两根指针，`p` 用于遍历链表，`minP` 指向当前遍历的最小一个节点的前一个结点；
- 当遍历结束时，当前 `minP->next` 指向的结点为最小结点

解答

```

void DeleteMin(LNode* &l)
{
    LNode* p = l;
    LNode* minP = l;

    while(p->next!=null)
    {
        if(p->next->data < minP->next->data)
        {
            minP = p;
        }
        p = p->next;
    }

    if(minP->next!=null)
    {
        LNode* q = minP->next;
        minP->next = q->next;
        free(q);
    }
}

```

```
}  
}
```

代码题 8

题干

- 题目：

8、试编写在不带头结点的单链表 L 中删除最小值点的高效算法（已知最小值唯一）

- 总结：

- 由于当前单链表不带头节点，因此需要分两次遍历进行删除；
- 第一步：找到最小节点的值；
- 第二步：删除给定值的结点

- 注意：

- 由于当前链表不带头节点，因此要单独判断下第一个结点为最小值的情况；

解答

```
void DeleteMin(LNode* &l)  
{  
    LNode* minP = l;  
    LNode* p = l;  
  
    while(p!=null)  
    {  
        if(p->data < minP->data)  
        {  
            minP->data = p->data;  
        }  
        p = p->next;  
    }  
  
    p=l;  
  
    if(l->data==minP->data)  
    {  
        l=l->next;  
        free(minP);  
        return;  
    }  
  
    while(p->next!=null)  
    {  
        if(p->next->data==min)  
        {  
            LNode* q = p->next;  
            p->next = q->next;  
        }  
    }  
}
```

```

        free(q);
    }
    p = p->next;
}
}

```

代码题 9

题干

- 题目：

9、给定一个单链表，按递增排序输出的单链表中各结点的数据元素，并释放节点所占空间

- 总结：

- 两层循环，内层循环从当前链表中找到最小的结点，并释放此结点。外层循环将此步骤重复，直到链表为空；

解答

```

void DeleteAndPrint(LNode* &l)
{
    LNode* p = l;
    LNode* minP = l;

    while(l->next!=null)
    {
        // 遍历链表，找到最小节点
        while(p->next!=null)
        {
            // 如果当前p结点的下一个结点的值比minP的下一个结点的值要小，则让minP指向当前
            // 的p结点
            if(p->next->data < minP->next->data)
            {
                minP = p;
            }

            p = p->next;
        }

        std::cout<<minP->next->data;

        // 释放当前找到的最小的结点
        LNode* q = minP->next;
        minP->next = q->next;
        free(q);

        // 继续从头开始找
        minP = l;
        p = l;
    }
}

```

```
}

    free(l);
}
```

代码题 10

题干

- 题目：

10、将一个带头节点的单链表 A 分解成两个带头节点的单链表 A 和 B，使 A 中含奇数位置元素，B 中含偶数位置元素，且相对位置不变

- 总结：

- 每两个元素是一对，两个元素中的第一个为奇元素，要留在 A 中；两个元素中的第二个为偶元素，要加入到 B 中；
- 因此需要定义三根指针，p 指针用于遍历整个链表，ra 指针用于指向一对元素中的第一个，rb 用于指向一对元素中的第二个；
- ra、rb 相当于 A、B 两个链表的链尾，利用链尾执行插入操作，将当前 p 结点插入到 A 和 B 中；

解答

```
// 返回值为B链表
LNode* SplitLink(LNode* &A)
{
    LNode* B = new LNode*;
    LNode* p = A->next;
    LNode* ra = A;
    LNode* rb = B;

    A->next = null;
    B->next = null;

    while(p!=null)
    {
        // 当前p处于一对元素中的第一个位置
        // 将当前p加入到A序列中
        ra->next = p;
        ra = p;

        // p进入到第二个位置
        p = p->next;

        // 当前p处于一对元素中的第二个位置
        rb ->next = p
        rb = p;
    }
}
```

```

        if(p!=null)
        {
            p = p->next;
        }
    }

    ra->next =null;

    return B;
}

```

代码题 11

题干

- 题目：

11、将一个单链表{a1,b1,a2,b2……an,bn}拆分成 { a1,a2……an }和{ bn,bn-1,……b1 }

- 总结：

- 和代码题 10 的思想类似，将链表的奇数列和偶数列分别放入链表 A、B 中，只不过对偶数列的插入使用的是头插法，保证在 B 中插入的是倒置的序列；

解答

```

LNode* SplitAndInvert(LNode* &A)
{
    LNode* B = new LNode*;
    B->next = null;

    // p 用于遍历整个链表
    LNode* p = A->next;

    // ra用于串联奇数链表
    LNode* ra = A;

    // 用来暂存要插入到B中的元素
    LNode* rb = B;

    while(p!=null)
    {
        ra->next = p;
        ra = p;
        p=p->next;

        // 当前如果是奇数列，则没有一对元素中的第二个元素，需要跳出循环
        if(p==null)

```

```

    {
        break;
    }

    // 存在第二个元素，将此元素插入B中
    rb = p;
    p=p->next;

    // rb现在代表要插入的元素，进行头插
    rb->next = B->next;
    B->next = rb;
}

ra->next = null;
return B;
}

```

代码题 12

题干

- 题目：

- **12、删除递增链表中重复的元素**

- 总结：

- 用两个指针遍历链表，顺序一前一后；
- 当 pre 的值和 p 的值不不同时，让 pre->next 直接指向 p->next，从而释放当前重复的 p 元素；

解答

```

void DeleteRepeatNumber(LNode* l)
{
    if(l==null)
    {
        return;
    }

    LNode* pre = l->next;
    LNode* p = null;

    // 因为是将当前pre和pre->next的值进行比较，所以需要确保pre->next存在
    while(pre->next!=null)
    {
        p = pre->next;

        if(pre->data==p->data)
        {
            pre->next = p->next;

```

```

        free(p);
    }
    else
    {
        pre = pre->next;
    }
}
}

```

代码题 13

题干

- 题目：
13、两个递增有序的单链表，设计算法成一个非递减有序的链表
- 总结：
 - 设置两根指针 `p、q`，分别对两个链表进行遍历；
 - 比较它们的大小，值小的放入到 A 链表中；

解答

```

void Merge(LNode* &A,LNode* &B)
{
    LNode* pa = A->next;
    LNode* pb = B->next;

    // p代表当前A链表的头元素
    LNode* p = A;
    A->next = null;

    while(pa!=null&&pb!=null)
    {
        // pa的元素更小，将pa结点插入到A结点中
        if(pa->data<pb->data)
        {
            p->next = pa;
            pa = pa->next;
            p = p->next;
        }
        else
        {
            p->next = pb;
            pb = pb->next;
            p = p->next;
        }
    }
    if(pa!=null)
    {

```

```

        p->next = pa;
    }
    if(pb!=null)
    {
        p->next = pb;
    }
}

```

代码题 14

题干

- 题目:

14、两个递增有序的单链表，设计算法成一个非递增有序的链表

- 总结:

- 非递增有序 = 递减有序;
- 方法一:
 - 可以将当前元素先按照递增有序插入链表中, 和代码题 13 一致;
 - 然后再对其进行逆置, 和代码题 5 一致;
- 方法二:
 - 在比较 ra 和 rb 的元素大小后, 便按照头插法将元素插入 A 中;
 - 然后再对余下的链表再进行头插;

解答: 方法一

```

void Merge(LNode* &A,LNode* &B)
{
    LNode* pa = A->next;
    LNode* pb = B->next;
    A->next = null;
    LNode* p = A;

    while(pa!=null&&pb!=null)
    {
        if(pa->data<pb->data)
        {
            p->next = pa;
            pa = pa->next;
            p = p->next;
        }
        else
        {
            p->next = pb;
            pb = pb->next;
            p = p->next;
        }
    }
}

```

```

    }
    if(pa!=null)
    {
        p->next = pa;
    }
    if(pb!=null)
    {
        p->next = pb;
    }

    p = A->next;
    A->next = null;
    while(p!=null)
    {
        LNode* q = p->next;
        p->next = A->next;
        A->next = p;
        p = q;
    }
}

```

解答：方法二

```

void Merge(LNode* &A,LNode* &B)
{
    LNode* pa = A->next;
    LNode* pb = B->next;
    A->next = null;
    LNode* p = A;

    while(pa!=null&&pb!=null)
    {
        if(pa->data<pb->data)
        {
            // 让p等于pa，然后对p进行头插
            p = pa;
            pa = pa->next;
            p->next = A->next;
            A->next = p;
        }
        else
        {
            // 让p等于pb，然后对p进行头插
            p = pb;
            pb = pb->next;
            p->next = A->next;
            A->next = p;
        }
    }
}

while(pa!=null)

```

```

{
    p = pa;
    pa = pa->next;
    p->next = A->next;
    A->next = p;
}

while(pb!=null)
{
    p = pb;
    pb = pb->next;
    p->next = A->next;
    A->next = p;
}
}

```

代码题 15

题干

- 题目：

15、A, B 两个单链表递增有序，从 A, B 中找出公共元素产生单链表 C，要求不破坏 A, B 结点

- 总结：

- pa、pb 指针用于遍历 A、B 链表，比较值的大小，相等时创建新节点，加入到 C 中；
- 并且谁小，谁往后移动一位；

解答

```

LNode* CommonElement(LNode* A, LNode* B)
{
    LNode* C = new LNode*;
    LNode* pc = C;

    LNode* pa = A->next;
    LNode* pb = B->next;

    while(pa!=null&&pb!=null)
    {
        if(pa->data==pb->data)
        {
            LNode* newNode = new LNode*;
            newNode->data = pa->data;
            pc->next = newNode;
            pc = pc->next;
            pa = pa->next;
            pb = pb->next;
        }
        else if(pa->data<pb->data)

```

```

        {
            pa = pa->next;
        }
        else if(pa->data>pb->data)
        {
            pb = pb->next;
        }
    }

    pc->next = null;
    return C;
}

```

代码题 16

题干

- 题目：

16、A, B 两个单链表递增有序，从 A, B 中找出公共元素并存放于 A 链表中

- 总结：

解答

```

void CommonElement(LNode* &A,LNode* &B)
{
    LNode* pa = A->next;
    LNode* pb = B->next;

    LNode* ra = A;
    A->next = null;

    LNode* tempNode;

    while(pa!=null&&pb!=null)
    {
        if(pa->data==pb->data)
        {
            // 把原本 A 链表中的pa结点加入到A当中
            ra->next = pa;
            ra=ra->next;
            pa = pa->next;

            //释放pb
            tempNode = pb;
            pb = pb->next;
            free(tempNode);
        }
    }
}

```

```

        else if(pa->data<pb->data)
        {
            tempNode = pa;
            pa = pa->next;
            free(tempNode);
        }
        else if(pa->data>pb->data)
        {
            tempNode = pb;
            pb = pb->next;
            free(tempNode);
        }
    }

    while(pa!=null)
    {
        tempNode = pa;
        pa = pa->next;
        free(tempNode);
    }

    while(pb!=null)
    {
        tempNode = pb;
        pb = pb->next;
        free(tempNode);
    }

    ra->next = null;
}

```

代码题 17

题干

- 题目：

17、两个序列分别为 A、B，将其存放到链表中，
判断 B 是否是 A 的连续子序列

- 总结：

- B 是待匹配序列，当匹配时，pa、pb 都往后移动一位；
- 当不匹配时，pb 要回到 B->next，即重新开始匹配。pa 要来到此轮匹配的位置的下一个位置，因此要用 prePa 来跟随 pa 指针，当 pa 指针开始和 pb 指针“并驾齐驱”的时候，prePa 留在原地。如果匹配失败，pre 往后移动一位；

解答

```

bool IsSubSeq(LNode* A,LNode* B)
{

```

```

LNode* pa = A->next;
LNode* pb = B->next;
LNode* prePa = pa;

while(pa!=null)
{
    if(pa->data==pb->data)
    {
        pa = pa->next;
        pb = pb->next;
    }
    else
    {
        pre = pre->next;
        pa = pre;
        pb = B->next;
    }
}

if(pb==null)
{
    // pb走完了B序列
    return true;
}
else
{
    return false;
}
}

```

代码题 18

题干

• 题目：

- 18、查找单链表中倒数第 k 个结点，若成功，
• 则输出该节点的 data，并返回 1，否则返回 0

• 总结：

- 使用双指针来遍历单链表。两个指针间的距离为 k，当 p 指针遍历完链表、到达 `null` 时，当前 q 指针的位置即为倒数第 k 个结点；
- 为了维持两指针的距离，需要先让 p 指针走 k 次。当 p 指针走了 k 次之后，p 和 q 一起后移；

解答

```

int ReturnLast(LNode* l,int k)
{
    LNode* p = l->next;

```

```

LNode* q = l;
int i = 1; // 两指针间的距离
while(p!=null)
{
    // 比如k=3,i=2时,i会自加到3,下一次循环时执行else,此时距离为3
    if(i<k)
    {
        i++;
        p = p->next;
    }
    else
    {
        p = p->next;
        q = q->next;
    }
}

// 如果当前链表只有k-1个元素,但却要返回第k个结点,则此时q指针一直不会移动,返回失败
if(q == l)
{
    return 0;
}
else
{
    std::cout<<q->data;
    return 1;
}
}

```

代码题 19

题干

- 题目:

19、用单链表保存 m 个整数,并且 $|data| \leq n$, 要求设计时间复杂度尽可能高效的算法,对于 $data$ 绝对值相等的点,仅保留第一次出现的点

41. (15 分) 用单链表保存 m 个整数, 结点的结构为: $[data][link]$, 且 $|data| \leq n$ (n 为正整数)。现要求设计一个时间复杂度尽可能高效的算法, 对于链表中 $data$ 的绝对值相等的结点, 仅保留第一次出现的结点而删除其余绝对值相等的结点。例如, 若给定的单链表 $head$ 如下:



则删除结点后的 $head$ 为:



要求:

- 1) 给出算法的基本设计思想。
- 2) 使用 C 或 C++ 语言, 给出单链表结点的数据类型定义。
- 3) 根据设计思想, 采用 C 或 C++ 语言描述算法, 关键之处给出注释。
- 4) 说明你所设计算法的时间复杂度和空间复杂度。

• 总结:

- 因为当前结点的绝对值小于等于 n , 因此当前结点的绝对值的可能取值为 $1 \sim n$, 一共有 n 个元素;
- 因此可以定义一个长度为 n 的 **虚拟数组**, 用于存储当前每一种取值的结点其出现的次数这一信息, 并且当访问到当前节点在虚拟数组中不为 0 时, 删除此结点。遍历完成后, 即可删除所有绝对值重复出现的结点;

• 注意:

- 定义虚拟数组后, 记得进行初始化 (讲所有元素置于 0) 以及释放 (`free`)

解答

```
void DeleteRepeat(LNode* &l)
{
    // index从1开始, 因此多定义一个位置
    int* isRepeat = new int[n+1];
    LNode* p = l;
    LNode* q = null;

    for(int i = 0; i < n+1; i++)
    {
        isRepeat[i] = 0;
    }

    while(q->next != null)
    {
        int data = q->next->data;
        if(data < 0)
        {
            data = -(data);
        }

        // 当前data第一次出现
        if(isRepeat[data] == 0)
        {
            // ... (rest of the code for deleting duplicates)
        }
    }
}
```

```

        isRepeat[data] = 1;
    }
    else
    {
        q = p->next;
        p->next = q->next;
        free(q);
    }
}

free(isRepeat);
}

```

代码题 20

题干

- 题目：
 - 20、判断带头结点的循环双链表是否对称
- 总结：
 - 两根指针，分两个方向同时走，判断它们值是否相等，如果相等则继续走，一直走到 `p->next==q` 或者是 `p==q` 时结束；

解答

```

bool IsSymmetry(LNode* l)
{
    LNode* p = l->next;
    LNode* q = l->prior;

    // p和q没走到一起
    while(p->next!=q&&p!=q)
    {
        if(p->data==q->data)
        {
            p = p->next;
            q = q->next;
        }
        else
        {
            return false;
        }
    }

    // 跳出循环时，两者data都相等，则返回真
    return true;
}

```

代码题 21

题干

- 题目:

21、有两个循环单链表，链表头指针分别为 h1,h2,试编写函数将 h2 链表接到 h1 之后,要求链接后仍保持循环链表形式

- 总结:

解答

```
void link(LNode* &h1, LNode* &h2)
{
    LNode *p, *q;
    p=h1,q=h2;
    while (p->next != h1)
    {
        p = p->next;
    }
    while(q->next != h2)
    {
        q= q->next;
    }

    p->next = h2;
    q->next = h1;
}
```

代码题 22

题干

- 题目:

22、设有一个带头结点的循环单链表，其结点值为正整数，设计算法反复找出链表内最小值并不断输出，并将结点从链表中删除，直到链表为空，再删除表头结点

- 总结:

- 和代码题 8 的思想一致;
- 两层循环，内层循环返回当前最小的结点，跳出循环后删除此节点。外层循环用于完成所有节点的删除;

解答

```

void DeleteMin(LNode* &l)
{
    LNode* p = l;
    LNode* minP = l;

    // 当头指针的next指向的是自己时，表示为空链表
    while(l != l->next)
    {
        while(p->next!=l)
        {
            if(p->next->data < minP->next->data)
            {
                minP = p;
            }

            p = p->next;
        }

        p = l->next;
        std::cout<<min->next->data<<" ";

        // 当前minP存储着最小元素结点
        LNode* q = min->next;
        min->next = q->next;
        free(q);
    }
    free(l);
}

```

代码题 23

题干

- 题目：
 - **23、判断单链表是否有环**
- 总结：
 - 使用双指针来判断是否成环；
 - 两根指针分别为 `fast` 和 `low`，`low` 走一步时，`fast` 走两步。当 `low==fast` 时，跳出遍历，并且表示有环。如果 `fast` 走到了 `null`，则表示无环；

解答

```

int IsLoop(LNode* l)
{
    LNode* fast = l;
    LNode* low = l;

    while(l!=null&&l->next!=null)
    {

```

```

        low = low->next;
        fast = fast->next->next;
        if(low == fast)
        {
            // 有环
            return 1;
        }
    }

    // 无环
    return 0;
}

```

代码题 24

题干

- 题目：

24、给定一个单链表 $L(a_1, a_2, a_3, \dots, a_n)$, 将其重新排列为 $(a_1, a_n, a_2, a_{n-1}, \dots)$

- 总结：

- 分成几个小步骤：
- 第一步：将单链表中间拆分为两个链表
 - 利用双指针（`low` 和 `fast`）遍历整个链表，`low` 走一步时，`fast` 走两步，当 `fast` 走到 `null` 时，此时 `low` 在链表的中间，将 `low` 及其左边的元素保留在 `L` 中，后续元素放入新链表 `L1` 中；
- 第二步：对 `L1` 链表进行逆置
- 第三步：将 `L1` 链表中结点逐个插入 `L` 中，

解答

```

void SplitAndInvert(LNode* &l)
{
    LNode* fast = l;
    LNode* low = l;
    while(fast!=null&&fast->next!=null)
    {
        fast = fast->next->next;
        low = low->next;
    }

    // 此时low指向中间元素,新建一个链表,存储low之后的元素
    LNode* newList = new LNode;
    newList->next = low->next;
    low->next = null;

    // 使用头插法,对newList进行逆置

```

```
LNode p = newList->next;
newList->next = null;

while(p!=null)
{
    LNode* q = p->next;
    p->next = newList->next;
    // 将当前链表头插到头部
    newList->next = p;
    p = q;
}

// 当前newList已完成逆置，接下来将其逐个插入链表l中
LNode* p2 = l->next;
LNode* p3 = newList->next;
newList->next = null;

while(p3!=null)
{
    LNode* nextP2 = p2->next;
    LNode* nextP3 = p3->next;

    p2->next = p3;
    p3->next = nextP2;
    p2 = nextP2;
    p3 = nextP3;
}

}
```

获取笔记持续中