链表知识点重点总结（掌握+背）

1. 基本用法

（1）结构体定义

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

声明一个单链表时，只需要声明一个头指针 `L`，指向单链表的第一个结点。

```
LNode *L; // 强调这是一个结点
```

```
LinkList L; // 强调这是一个单链表
```

（2） 按位查找

```
// 按位查找：返回第 i 个元素，带头结点，头结点看作是第 0 个结点
LNode *GetElem(LinkList L, int i)
{
    if (i < 0){
        return NULL;
    }
    LNode *p;
    int j = 0;                  // 当前 p 指向的是第几个结点
    p = L;                      // 指向第 0 个结点
    while (p != NULL && j < i) // 循环找到第 i-1 个结点
    {
        p = p->next;
        j++;
    }
    return p;
}
```

（3） 按值查找

```
// 按值查找，找到数据域 == e 的结点
int LocateElem(LinkList L, int e){
    LNode *p = L->next;
    // 从第 1 个结点开始查找数据域为 e 的结点
    while (p != NULL && p->data != e)
    {
        p = p->next;
    }
    // 找到后返回该结点指针，否则返回 NULL
    return p;
}
```

(4) 求表长度

```
// 求表的长度
int Length(LinkList L){
    int len = 0;
    LNode *p = L;
    while (p->next != NULL)
    {
        p = p->next;
        len++;
    }
    return len;
}
```

(5) 删除值为x的节点

```
// 删除值为x的节点的函数
void DeleteX(LinkList L, int x) {
    LNode *p = L->next; // p指向首元节点
    LNode *pre = L;     // pre指向头节点
    while (p != NULL) {
        if (p->data == x) { // 找到值为x的节点
            pre->next = p->next; // 删除操作
            free(p); // 释放内存
            p = pre->next; // 移动p到下一个节点
        } else {
            pre = p; // pre指向当前节点
            p = p->next; // p指向下一个节点
        }
    }
}
```

(6) 删除序号为k的节点

```
// 删除序号为k的节点的函数
bool DeleteKthNode(LinkList L, int k) {
    if (k <= 0) return false; // k不合法
    LNode *p = L; // p指向头结点
    int i = 0; // 计数器
    // 寻找第k-1个节点
    while (p != NULL && i < k - 1) {
        p = p->next;
        i++;
    }
    // 如果p为NULL或p的下一个节点为NULL，说明不存在第k个节点
    if (p == NULL || p->next == NULL) return false;
    LNode *q = p->next; // q指向第k个节点
    p->next = q->next; // 删除第k个节点
    free(q); // 释放第k个节点的内存
    return true; // 删除成功
}
```

(7) 在结点 p 前插入元素 e

```
// 前插操作：在结点 p 前插入元素 e
bool InsertPriorNode(LNode *p, int e){
    if (p == NULL)
    {
        return false;
    }
    LNode *s = (LNode *)malloc(sizeof(LNode));
    s->next = p->next;
    p->next = s;
    s->data = p->data;
    p->data = e;
    return true;
}
```

(8) 按位序插入

```
// 在第 i 个位置插入元素 e，带头结点
bool ListInsert(LinkList &L, int i, int e)
{
    // i 的值必须是合法的位序
    if (i < 1)
    {
        return false;
    }
    LNode *p;
    int j = 0;                      // 当前 p 指向的是第几个结点
    p = L;                          // 指向第 0 个结点
    while (p != NULL && j < i - 1) // 循环找到第 i-1 个结点
    {
        p = p->next;
        j++;
    }
    if (p == NULL)
    {
        return false;
    }
    LNode *s = (LNode *)malloc(sizeof(LNode));
    s->data = e;
    s->next = p->next;
    p->next = s;
    return true;
}
```

(9) 尾插法构建单链表

```
// 尾插法建立单链表,带头结点
LinkList List_TailInsert(LinkList L){
    int x;
    L = (LNode *)malloc(sizeof(LNode));
    LNode *s, *r = L; // r 为表尾指针
    scanf("%d", &x);
```

```
    while (x != 9999)
    {
        s = (LNode *)malloc(sizeof(LNode));
        s->data = x;
        r->next = s;
        r = s; // r 指向新的表尾结点
        scanf("%d", &x);
    }
    r->next = NULL;
    return L;
}
```

（10）头插法构建单链表（可实现链表逆置）

```
// 头插法建立单链表,带头结点
LinkList List_HeadInsert(LinkList L)
{
    int x;
    L = (LNode *)malloc(sizeof(LNode));
    L->next = NULL; // 一定要初始化头结点的 next
    LNode *s;
    scanf("%d", &x);
    while (x != 9999)
    {
        s = (LNode *)malloc(sizeof(LNode));
        s->data = x;
        s->next = L->next;
        L->next = s;
        scanf("%d", &x);
    }
    return L;
}
```

2. 建立单链表（均采用尾插法，头插法照着修改即可）

（1）从键盘中获取数据建立单链表

```
// 尾插法建立单链表,带头结点
LinkList List_TailInsert(LinkList L){
    int x;
    L = (LNode *)malloc(sizeof(LNode));
    LNode *s, *r = L; // r 为表尾指针
    scanf("%d", &x);
    while (x != 9999)
    {
        s = (LNode *)malloc(sizeof(LNode));
        s->data = x;
        r->next = s;
        r = s; // r 指向新的表尾结点
        scanf("%d", &x);
    }
    r->next = NULL;
```

```
    return L;
}
```

(2) 从数组中建立单链表

```
// 创建带头结点的单链表
LinkList CreateList(int arr[], int n) {
    LinkList L = (LinkList)malloc(sizeof(LNode));
    L->next = NULL;
    for (int i = 0; i < n; i++) { // 尾插法建立单链表
        LNode *s = (LNode *)malloc(sizeof(LNode));
        s->data = arr[i];
        s->next = L->next;
        L->next = s;
    }
    return L;
}
```

(3) 从文件中获取数据建立单链表

```
//从文件中读取数据建立单链表
LinkList CreateList(char *filename) {
    FILE *fp = fopen(filename, "r"); // 打开文件用于读取
    LinkList L = (LinkList)malloc(sizeof(LNode)); // 创建头结点
    L->next = NULL;
    LNode *r = L; // r始终指向终端结点
    int value;
    while (1) {
        // 从文件中读取整数（读取其他数据，修改一下即可）
        if(fscanf(fp, "%d", &value) == -1){//如果读取失败，直接跳出循环。
            break;
        }
        LNode *s = (LNode *)malloc(sizeof(LNode));
        s->data = value;
        s->next = NULL;
        r->next = s; // 将s插入到终端结点之后
        r = s; // r指向新的终端结点
    }

    fclose(fp); // 关闭文件
    return L;
}
```

3. 对链表进行排序（掌握一种即可）

(1) 插入排序（重点掌握，推荐这个）

```
// 插入排序
void SortList(LinkList L) {
    LNode *L1 = L->next; // L1指向第一个结点
    L->next = NULL; // 初始化排序好的链表为空
```

```
    while (L1) {
        LNode *next = L1->next; // 保存下一个结点的地址
        LNode *p = *L;
        // 找到插入位置
        while (p->next && p->next->data < L1->data) {
            p = p->next;
        }
        L1->next = p->next; // 插入L1结点
        p->next = L1;
        L1 = next; // 处理下一个结点
    }
}
```

(2) 冒泡排序

```
void BubbleSort_Node(LinkList head){
     LNode *pre,*p,*q;
     LNode *tail = NULL;              //尾指针，每次冒泡排序之后，最后一个元素都已经确定顺序
    while(head->next!= tail){             //冒泡次数
        pre = head;
        p = head->next;
        q = p->next;
        while(p->next!=tail){             //冒泡长度
            if(p->data > q->data){        //如果前一个数比后一个数大，交换指针
                pre->next = q;
                p->next = q->next;
                q->next =  p;
            }
            else{
                p = p->next;          //不用交换指针，继续往下找
            }
            pre = pre->next;          //p指针已经在上面的if和else中移动了，无需再移动
            q = p->next;
        }
        tail = p;                     //经过一次冒泡，当前最后一个元素已经排好序，tail前移
    }
}
```

4. 单链表删除重复元素

```
// 删除单链表中重复元素的函数,带头结点
void DeleteRepeat(LinkList L) {
    LNode *p = L->next; // p指向首元结点
    while (p != NULL) {
        LNode *q = p->next; // q指向p的下一个结点
        LNode *pre = p; // pre指向p的前一个结点
        while (q != NULL) {
            if (q->data == p->data) { // 如果q的值与p的值相同
                pre->next = q->next; // 删除q结点
                free(q); // 释放q结点的内存
                q = pre->next; // q指向新的下一个结点
            } else {
```

```
                pre = q; // pre指向q
                q = q->next; // q指向下一个结点
            }
        }
        p = p->next; // p指向下一个结点
    }
}
```

5. 合并两个有序链表

```
// 合并两个有序链表的函数
LinkList Merge(LinkList La, LinkList Lb) {
    LinkList Lc = (LinkList)malloc(sizeof(LNode)); // 创建头结点
    Lc->next = NULL;
    LNode *pa = La->next; // pa指向La的首元结点
    LNode *pb = Lb->next; // pb指向Lb的首元结点
    LNode *pc = Lc; // pc指向Lc的终端结点

    while (pa != NULL && pb != NULL) {
        if (pa->data <= pb->data) { // 如果La的当前元素小于等于Lb的当前元素
            pc->next = pa; // 将pa插入到Lc的终端结点之后
            pa = pa->next; // pa指向下一个结点
            pc = pc->next; // 更新Lc的终端结点
        } else { // 如果Lb的当前元素小于La的当前元素
            pc->next = pb; // 将pb插入到Lc的终端结点之后
            pb = pb->next; // pb指向下一个结点
            pc = pc->next; // 更新Lc的终端结点
        }
    }
    // 将剩余的元素连接到Lc的末尾
    pc->next = (pa != NULL ? pa : pb);

    return Lc;
}
```

6. 拆分链表，按照奇数偶数（性别等）

```
// 拆分链表为奇数链表和偶数链表的函数
void SplitList(LinkList L, LinkList L1, LinkList L2) {
    L1 = (LinkList)malloc(sizeof(LNode)); // 创建奇数链表的头结点
    L2 = (LinkList)malloc(sizeof(LNode)); // 创建偶数链表的头结点
    LNode *p = L->next; // p指向首结点
    LNode *r1 = L1, *r2 = L2; // r1和r2分别指向奇数链表和偶数链表的终端结点

    while (p != NULL) {
        if (p->data % 2 == 0) { // 如果当前结点的数据是偶数
            r2->next = p; // 将p插入到偶数链表的终端结点之后
            r2 = p; // 更新偶数链表的终端结点
            p = p->next; // p指向下一个结点
        } else { // 如果当前结点的数据是奇数
```

```
            r1->next = p; // 将p插入到奇数链表的终端结点之后
            r1 = p; // 更新奇数链表的终端结点
            p = p->next; // p指向下一个结点
        }
    }
    r1->next = NULL; // 奇数链表的末尾置为NULL
    r2->next = NULL; // 偶数链表的末尾置为NULL
}
```