

数据结构实验报告——实验四

学号： 20201060330 姓名： 胡诚皓 得分： _____

一、实验目的

1. 复习线性表的逻辑结构、存储结构及基本操作；
2. 掌握顺序表和（带头结点）单链表；
3. 了解有序表。

二、实验内容

1. （必做题）有序线性表的构造与合并

假设有有序表中数据元素类型是整型，请采用顺序表或（带头结点）单链表实现：

(1) OrderInsert(&L, e, int (*compare)(a, b))

//根据有序判定函数 compare，在有序表 L 的适当位置插入元素 e；

(2) OrderInput(&L, int (*compare)(a, b))

//根据有序判定函数 compare，并利用有序插入函数 OrderInsert，构造有序表 L；

(3) OrderMerge(&La, &Lb, &Lc, int (*compare)())

//根据有序判定函数 compare，将两个有序表 La 和 Lb 归并为一个有序表 Lc。

2. （必做题）多项式的构造与相加

请实现：

- (1) 升幂多项式的构造，升幂多项式是指多项式的各项按指数升序有序，约定系数不能等于 0，指数不能小于 0；
- (2) 两个升幂多项式的相加。

3. （选做题）约瑟夫环

问题描述：将数字 1,2, ..., n 环形排列；按顺时针方向从 1 开始计数，计满 k 时输出该位置上的数字，同时从环中删除该数字；然后从下一位置开始重新开始计数，直到环中所有数均被输出为止。

请使用顺序表或链表实现：对输入的任意 n 和 k，输出相应的出列序。

三、数据结构及算法描述

1. （必做题）有序线性表的构造与合并

(1) 数据结构

此题使用了带头结点的链式存储的线性表，Lnode 结构体作为每个结点，Link 为指向 Lnode 的指针类型，其中的数据域 data 以 int 型为例，指针域 next 为指向下一个结点的指针。宏定义了实际为 int 型的 Status 类型，OK（实际为 1）与 ERROR（实际为 0）作为 Status 类型的值。

其中各个函数参数中的 int (*compare)(Lnode a, Lnode b) 函数指针都代表“若 compare 的返回值不为 0 时，认为 a 应该在 b 的前面”其中的 a、b 都为 Lnode 类型的形参。不单纯使用返回比较两个 int 值的函数指针目的在于，当 Lnode 中的数据域的定义不只是 int 时，各个有 compare 函数指针的函数更具普适性。

(2) 算法描述

程序中有三个辅助函数 Lnode *createNode(int data)、Lnode *deepCpy(const Lnode *src)、void displayLink(Link head)，分别用于创建以 data 为数据域的结点并返回指向这个新创建的结点的指针、深拷贝 src 指向的结点元素并返回指向拷贝出来的新结点的指针、输出以 head 为头结点的整个链表。

OrderInsert(&L, e, int (*compare)(a, b))

- ① 判断 L 是否为只有头结点的空链表，若为空链表，则直接将新的结点 e 放到链表中；
- ② 由于之前每次添加结点时都保证以 compare 的规则进行排列，因此只要保证这一次将要添加的结点放在合适的位置即可。从头往尾遍历链表，直到找到某个结点，这个结点的下一个结点按照 compare 规则应该位于要添加的结点的后面。那么，要添加的结点的合适位置就在这个遍历到的结点的下一个，将要添加的结点插入到遍历到的结点的后面；
- ③ 若没有遍历到符合②中条件的结点，说明链表中所有已有结点按照 compare 规则都应该处于要添加的结点的前面，就把要添加的结点放在链表的最后。

OrderInput(&L, int (*compare)(a, b))

- ① 让用户输入要添加结点个数；

- ② 让用户依次输入要添加的结点的数据域中 int 型的 data，先调用 createNode 创建新结点，再调用 OrderInsert 将新结点插入到链式存储结构有序表的合适位置。

OrderMerge(&La, &Lb, &Lc, int (*compare)())

- ① 判断 La、Lb 是否为空，若有为空的直接返回 ERROR；判断 Lc 是否为空，因为要将合并后的有序表存在以 Lc 为头结点的链表中，Lc 不为空直接返回 ERROR
- ② 从头开始分别遍历 La、Lb 并按照 compare 规则将 La、Lb 中的各个元素分别放入 Lc 中。因为是从头往尾遍历 La、Lb，先要插入 Lc 的是一定是靠前的元素，所以使用尾插法向 Lc 中插入结点。为了不影响 La、Lb 本身的构造，调用 deepCpy 拷贝各个要加入 Lc 的结点，将深拷贝得到的结点插入 Lc 中。
- ③ 将 La、Lb 中剩余的未加入 Lc 中的结点元素都加入 Lc 中

2. （必做题）多项式的构造与相加

(1) 数据结构

此题使用 Lnode 作为存储多项式的某一项，Lnode 中的 coefficient 存储该项的系数、frequency 存储该项的次数，Link 为 Lnode 类型的指针。宏定义了实际为 int 型的 Status 类型，OK（实际为 1）与 ERROR（实际为 0）作为 Status 类型的值。

其中各个函数参数中的 int (*compare)(Lnode a, Lnode b) 函数指针都代表“若 compare 的返回值不为 0 时，认为 a 应该在 b 的前面”其中的 a、b 都为 Lnode 类型的形参。

本题中按照次数升序排列各项，byAscending 中就是按照 Lnode a, Lnode b 的 frequency 数据域进行比较，当 a.frequency < b.frequency 时，byAscending 返回值不为零，意味着 a 应该在 b 的前面。

(2) 算法描述

为了方便编写其他功能，在此题中编写了 Lnode *createNode(int, int)、Lnode *deepCpy(const Lnode *)、void displayPoly(Link)三个辅助函数，分别用于根据两个参数创建某一项结点并返回指向多项式某一项结点的指针、深拷贝一个结点并返回一个指向拷贝出来的新结点的指针、输出多项式。同时继续沿用上一题中将结点插入链表的函数 Status OrderInsert(Link, Lnode *, int (*)(Lnode, Lnode))

Status createPoly(Link head, int num, ...)

创建以 head 为头结点的多项式链表，num 为想要创建的多项式的项数，之后为一个可变长度的参数列表，应为系数 1、次数 1、系数 2、次数 2.....、系数 num、次数 num。

- ① 定义变量 tmp 用于在循环中暂时存储将要加入链表的结点，定义并初始化可变长度参数列表 coeAndFre
- ② 每次循环都将 num-1，一直循环到 num 为 0，从可变长度参数列表中读取系数和次数，调用 createNode 创建新结点并用 OrderInsert 将新创建的结点插入链表
- ③ 结束可变长度参数列表 coeAndFre 的使用

Status addPloy(Link polyA, Link polyB, Link polyC, int (*compare)(Lnode a, Lnode b))

将以 polyA、polyB 为头结点的多项式相加，并把结果存储在以 polyC 为头结点的多项式链表中，使用 compare 函数作为各项前后顺序比较的规则

- ① polyA、polyB 不能为空，polyC 必须为空，不满足该条件直接返回 ERROR
- ② 使 pA、pB 分别指向 polyA 与 polyB 的第一个实际结点元素。
- ③ 由于多项式是根据 compare 规则从前往后排列的，若根据 compare 规则，pA 所指向的结点应处于较前位置，则说明 polyB 中没有对应的项，也就无需进行相加操作，将结点深拷贝后直接加入 polyC 中，同时 pA 向后移试图在 polyA 中寻找与 pB 对应的项。
- ④ 对于 pB 所指向的结点应处于较前位置的情况也是相同的。若根据 compare 规则，pB 所指向的结点应处于较前位置，则说明 polyA 中没有对应的项，也就无需进行相加操作，将结点深拷贝后直接加入 polyC 中，同时 pB 向后移试图在 polyB 中寻找与 pA 对应的项。
- ⑤ 若 pA 与 pB 根据 compare 规则是相对应的项，就调用 createNode 创建新结点，新结点的系数为这两个项的系数之和，次数不变。
- ⑥ 将 polyA 或 polyB 中剩余的结点做深拷贝后加入 polyC 中

3. （选做题）约瑟夫环

（1）数据结构

使用循环链表来存储本题的结点，使用的是没有头结点的循环链表，即在第一个结点中直接存数据，本题在 solve 的循环中使用 cur 来指向当前判断的结点，pre 为 cur 的前一个结点，目的是为了当 cur 是需要删除的结点时，可以直接让 pre->next=cur->next

（2）算法描述

Status createLoopLink (Link head, int n)

用来创建以 head 为第一个结点的有 n 个人的约瑟夫环，需要一个 rear 指针，始终指向目前有数据的最后一个结点，在插入新结点时直接 rear->next = tmp 即可。

- ① 初始化局部变量，使得 head->data = 1, rear=head
- ② i 从 2 循环到 n，创建各个结点并接入链表
- ③ 将 rear->next 赋值为 head，从而使得约瑟夫环闭合，形成循环链表

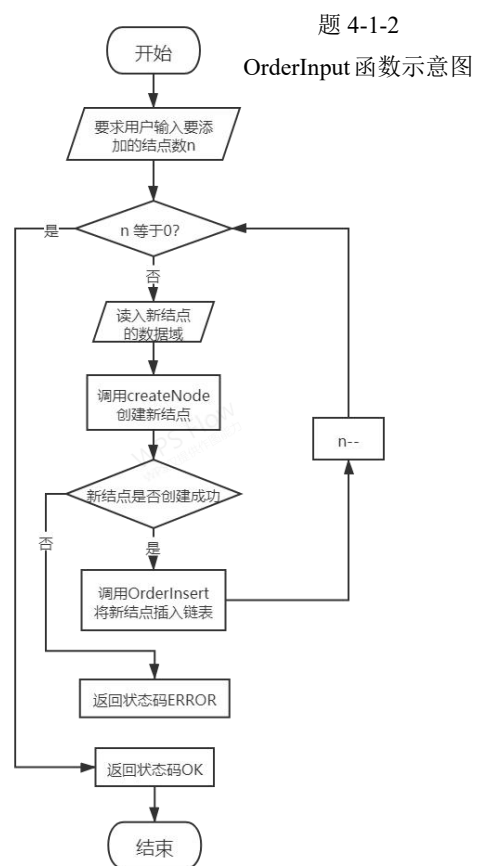
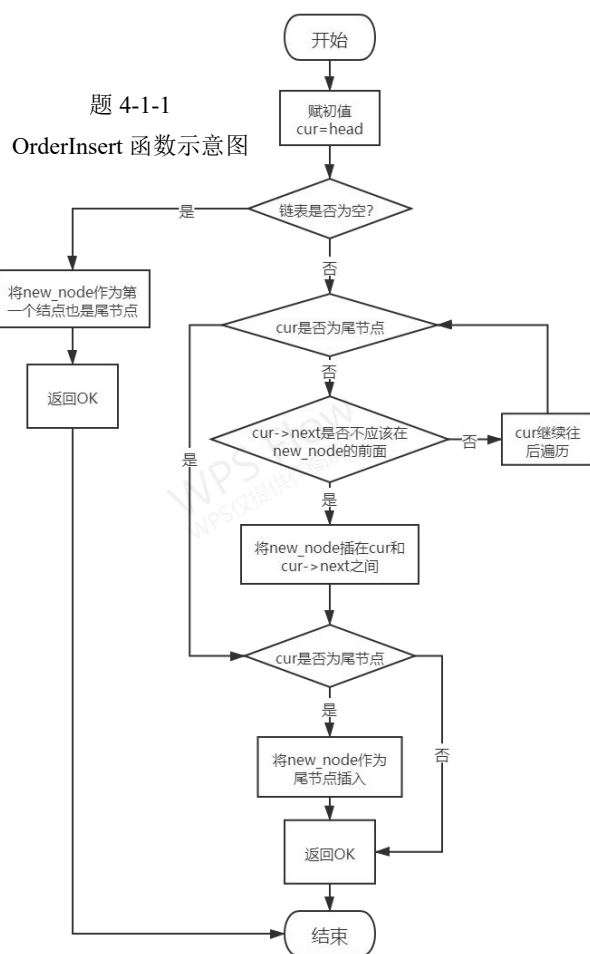
Status solve(Link head, int k)

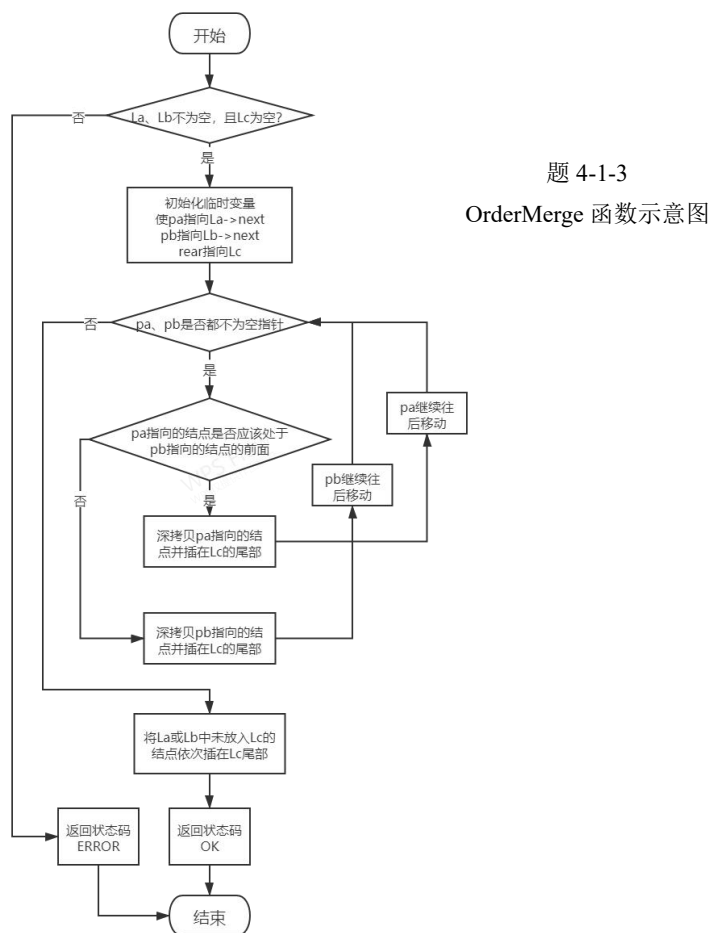
用来解决题目描述的约瑟夫环的问题，由于 head 是一个循环链表，先找到 head 的前一个结点，也就是这个循环链表的“尾结点”，局部变量 int count 作为计数的临时变量，不断循环删除第 k 个结点。

- ① 初始化局部变量，使得 cur=head, pre=head
- ② 不断循环 cur，一边循环一边把 cur 往后移，直到 cur 的下一个元素是自己，也就是循环链表中只剩下一个元素的情况。
- ③ 将 count++，在使让 count 对 k 取余，若取余后为 0，说明已经到了第 k 个，直接删除此时 cur 指向的结点。否则就直接往后循环，将 pre、cur 各自往后推。

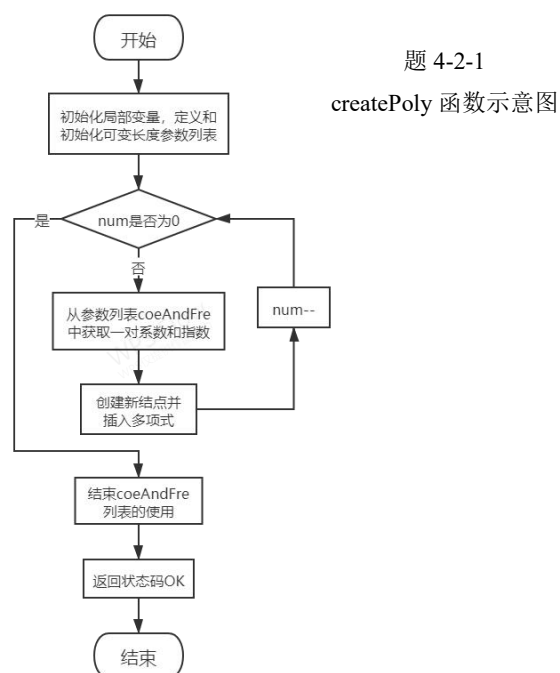
四、详细设计

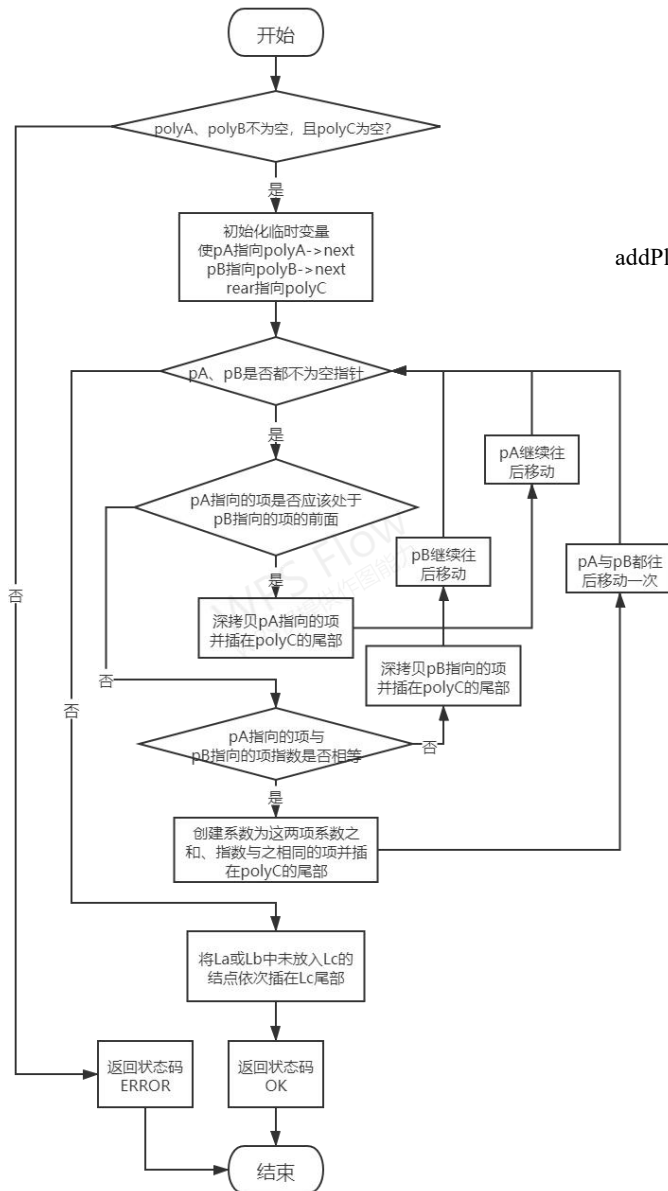
1. （必做题）有序线性表的构造与合并





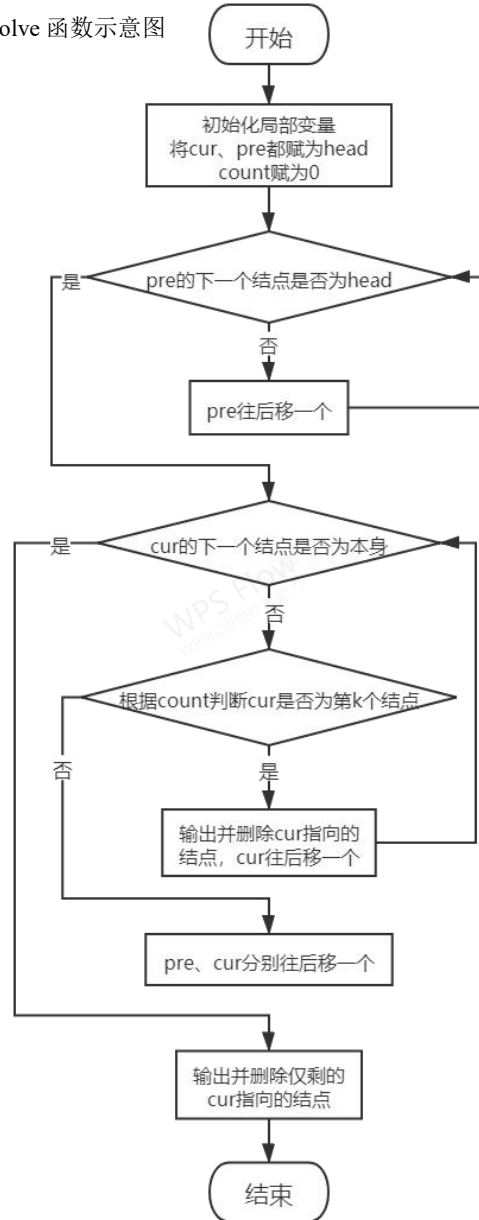
2. (必做题) 多项式的构造与相加



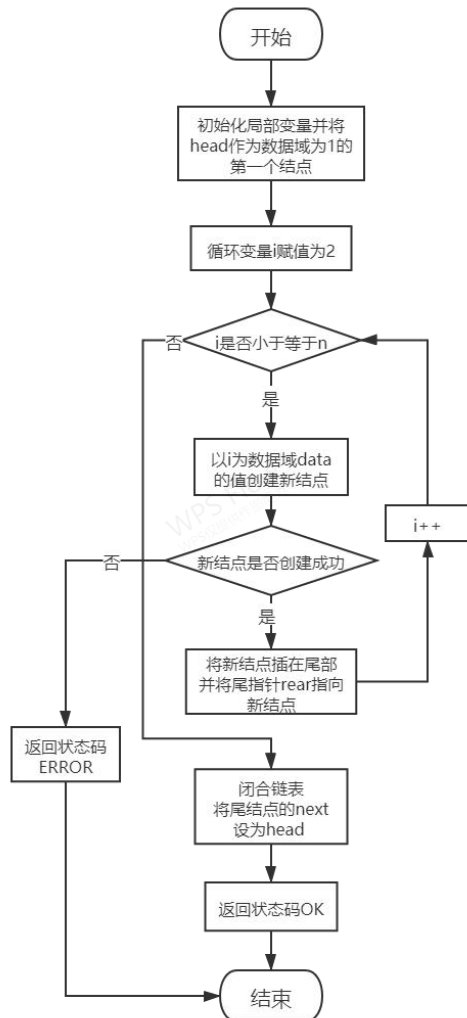


3. （选做题）约瑟夫环

题 4-3-2
solve 函数示意图



题 4-3-1
createLoopLink 函数示意图



五、程序代码

1. （必做题）有序线性表的构造与合并



4-1.c

```

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

#define Status int
#define OK 1
  
```

```

#define ERROR 0

typedef struct Lnode {
    int data;
    struct Lnode *next;
} Lnode, *Link;

Status OrderInsert(Link, Lnode *, int (*)(Lnode, Lnode));
Status OrderInput(Link head, int (*)(Lnode, Lnode));
Status OrderMerge(Link, Link, Link, int (*)(Lnode, Lnode));
Status OrderSort(Link head, int (*)(Lnode, Lnode));

Lnode *createNode(int);
Lnode *deepCpy(const Lnode *);
int byAscending(Lnode, Lnode);
void displayLink(Link);

int main() {
    Link linkList1, linkList2;
    Link mergeList;
    Lnode *tmp;

    linkList1 = createNode(-1);
    linkList2 = createNode(-1);
    mergeList = createNode(-1);
    printf("Create linkList1\n");
    OrderInput(linkList1, byAscending);
    printf("Create linkList2\n");
    OrderInput(linkList2, byAscending);

    printf("linkList1: ");
    displayLink(linkList1);
    printf("linkList2: ");
    displayLink(linkList2);

    OrderMerge(linkList1, linkList2, mergeList, byAscending);
    printf("merged linklist1&2: ");
    displayLink(mergeList);

    printf("\n");
    system("pause");

    return 0;
}

```

```

//若 compare 的返回值不为 0, 认为 a 应该在 b 的前面
Status OrderInsert(Link head, Lnode *new_node, int (*compare)(Lnode a, Lnode b)) {
    Link cur = head;

    if (head->next == NULL) {
        new_node->next = NULL;
        head->next = new_node;
        return OK;
    }

    while (cur->next != NULL) {
        if (!compare(*(cur->next), *new_node)) {
            new_node->next = cur->next;
            cur->next = new_node;
            break;
        }
        cur = cur->next;
    }
    if (cur->next == NULL) {
        cur->next = new_node;
        new_node->next = NULL;
    }

    return OK;
}

Status OrderInput(Link head, int (*compare)(Lnode, Lnode)) {
    int n, tmp_data;
    Lnode *tmp;
    printf("Input the number of nodes to add:\n");
    scanf("%d", &n);
    printf("Input integer data to save in each node\n");
    printf("(each number ends with 'Enter' key or input in one line separating with
space):\n");
    while (n-->0) {
        scanf("%d", &tmp_data);
        if ((tmp = createNode(tmp_data)) == ERROR)
            return ERROR;
        OrderInsert(head, tmp, compare);
    }
    return OK;
}

```

```

//若 compare 的返回值不为 0，认为 a 应该在 b 的前面
Status OrderSort(Link head, int (*compare)(Lnode a, Lnode b)) {
    //需要调整的链表是空的，直接返回 ERROR
    if (head->next == NULL)
        return ERROR;
    //p 始终指向链表中还未调整位置的第一个结点
    Lnode *p = head->next->next, *tmp;
    head->next->next = NULL;

    while (p) {
        tmp = p->next;
        OrderInsert(head, p, compare);
        p = tmp;
    }

    return OK;
}

//若 compare 的返回值不为 0，认为 a 应该在 b 的前面
//La 与 Lb 必须是与将要产生的 Lc 一样，按照同一种 compare 方法构造的有序表
Status OrderMerge(Link La, Link Lb, Link Lc, int (*compare)(Lnode a, Lnode b)) {
    //Lc 作为 La 和 Lb 合并后的链表的头结点，一开始需要是空的
    //La、Lb 也都不能是空的，不然没有合并的意义
    if (Lc->next != NULL || La->next == NULL || Lb->next == NULL)
        return ERROR;

    //使用尾插法构造 Lc
    Lnode *pa = La->next, *pb = Lb->next, *rear = Lc;
    Lnode *tmp;
    while (pa != NULL && pb != NULL) {
        //pa 应该放前面
        if (compare(*pa, *pb)) {
            //深拷贝结点加入 Lc 中，防止 La、Lb 中的结点受到影响被改变
            tmp = deepCpy(pa);
            rear->next = tmp;
            rear = tmp;
            pa = pa->next;
            rear->next = NULL;
        } else {
            tmp = deepCpy(pb);
            rear->next = tmp;
            rear = tmp;
            pb = pb->next;
            rear->next = NULL;
        }
    }
}

```

```

    }

    //剩余元素放入
    while (pa != NULL) {
        tmp = deepCpy(pa);
        rear->next = tmp;
        rear = tmp;
        pa = pa->next;
        rear->next = NULL;
    }
    while (pb != NULL) {
        tmp = deepCpy(pb);
        rear->next = tmp;
        rear = tmp;
        pb = pb->next;
        rear->next = NULL;
    }

    return OK;
}

int byAscending(Lnode a, Lnode b) {
    return a.data < b.data;
}

Lnode *createNode(int data) {
    Link p;
    p = (Link) malloc(sizeof(Lnode));
    p->data = data;
    p->next = NULL;
    return p;
}

void displayLink(Link head) {
    if (head->next == NULL)
        return;
    Link p = head->next;
    printf("%d", p->data);
    p = p->next;
    while (p) {
        printf("->%d", p->data);
        p = p->next;
    }
    printf("\n");
}

```

```

}

//返回对 src 结点的一份深拷贝
Lnode *deepCpy(const Lnode *src) {
    Lnode *res;
    res = (Lnode *) malloc(sizeof(Lnode));
    res = (Lnode *) memcpy(res, src, sizeof(Lnode));

    return res;
}

```

2. （必做题）多项式的构造与相加



4-2.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>

#define Status int
#define OK 1
#define ERROR 0

typedef struct Lnode {
    //存储系数
    int coefficient;
    //存储次数
    int frequency;
    struct Lnode *next;
} Lnode, *Link;

Status OrderInsert(Link, Lnode *, int (*)(Lnode, Lnode));
Status createPoly(Link head, int num, ...);
Status addPoly(Link, Link, Link, int (*)(Lnode, Lnode));

Lnode *createNode(int, int);
Lnode *deepCpy(const Lnode *);
int byAscending(Lnode, Lnode);
void displayPoly(Link);

int main() {
    Link poly1, poly2, poly3;

```

```

    poly1 = createNode(-1, -1);
    poly2 = createNode(-1, -1);
    poly3 = createNode(-1, -1);
    createPoly(poly1, 2, 1, 2, 5, 6);
    createPoly(poly2, 3, 1, 2, 1, 1, 1, 3);
    printf(" ");
    displayPoly(poly1);
    printf("+");
    displayPoly(poly2);
    printf("=");

    addPoly(poly1, poly2, poly3, byAscending);
    displayPoly(poly3);

    printf("\n");
    system("pause");
    return 0;
}

//创建多项式，使用以 head 为头结点的链表存储
//num 为要创建的多项式的项数
//后面的参数依次为对应的系数、次数的交替
Status createPoly(Link head, int num, ...) {
    Lnode *tmp;
    int tmpCoe, tmpFre;
    //定义可变长度数组，用来接收多项式的系数和次数
    va_list coeAndFre;
    //初始化参数列表
    va_start(coeAndFre, num);

    while (num--) {
        tmpCoe = va_arg(coeAndFre, int);
        tmpFre = va_arg(coeAndFre, int);
        tmp = createNode(tmpCoe, tmpFre);
        OrderInsert(head, tmp, byAscending);
    }

    va_end(coeAndFre);

    return OK;
}

//将 polyA 与 polyB 两个多项式相加的结果存在 polyC 中

```

```

//polyA、polyB 不能为空，polyC 必须为空
//若 compare 的返回值不为 0，认为 a 应该在 b 的前面
Status addPoly(Link polyA, Link polyB, Link polyC, int (*compare)(Lnode a, Lnode b))
{
    if (polyA->next == NULL || polyB->next == NULL || polyC->next != NULL)
        return ERROR;
    Link pA = polyA->next, pB = polyB->next;
    //使用尾插法构造 Lc
    Link rear = polyC;
    Lnode *tmp;

    while (pA != NULL && pB != NULL) {
        //pA 必处于较前位置，说明 polyB 中没有对应项，直接加入 polyC 中
        if (compare(*pA, *pB)) {
            //深拷贝结点
            tmp = deepCpy(pA);
            rear->next = tmp;
            rear = tmp;
            pA = pA->next;
            rear->next = NULL;
        }
        //pA 与 pB 次数相等，相加后加入 polyC
        else if (pA->frequency == pB->frequency) {
            tmp = createNode(pA->coefficient + pB->coefficient, pA->frequency);
            rear->next = tmp;
            rear = tmp;
            pA = pA->next;
            pB = pB->next;
            rear->next = NULL;
        }
        //pB 必处于较前位置，说明 polyA 中没有对应项，直接加入 polyC 中
        else {
            tmp = deepCpy(pB);
            rear->next = tmp;
            rear = tmp;
            pB = pB->next;
            rear->next = NULL;
        }
    }

    //剩余元素放入
    while (pA != NULL) {
        tmp = deepCpy(pA);
        rear->next = tmp;
    }
}

```



```

        rear = tmp;
        pA = pA->next;
        rear->next = NULL;
    }
    while (pB != NULL) {
        tmp = deepCpy(pB);
        rear->next = tmp;
        rear = tmp;
        pB = pB->next;
        rear->next = NULL;
    }

    return OK;
}

```

//若 compare 的返回值不为 0，认为 a 应该在 b 的前面

```

Status OrderInsert(Link head, Lnode *new_node, int (*compare)(Lnode a, Lnode b)) {
    Link cur = head;

    if (head->next == NULL) {
        new_node->next = NULL;
        head->next = new_node;
        return OK;
    }

    while (cur->next != NULL) {
        if (!compare(*(cur->next), *new_node)) {
            new_node->next = cur->next;
            cur->next = new_node;
            break;
        }
        cur = cur->next;
    }
    if (cur->next == NULL) {
        cur->next = new_node;
        new_node->next = NULL;
    }

    return OK;
}

```

//若 compare 的返回值不为 0，认为 a 应该在 b 的前面

//若 compare 的返回值不为 0，认为 a 应该在 b 的前面

//La 与 Lb 必须是与将要产生的 Lc 一样，按照同一种 compare 方法构造的有序表

```
int byAscending(Lnode a, Lnode b) {
    return a.frequency < b.frequency;
}

Lnode *createNode(int coe, int fre) {
    Link p;
    p = (Link) malloc(sizeof(Lnode));
    p->coefficient = coe;
    p->frequency = fre;
    p->next = NULL;
    return p;
}

void displayPoly(Link head) {
    //空链表，无需输出，直接返回
    if (head->next == NULL)
        return;
    int tmpCoe, tmpFre;
    Link p = head->next;
    tmpCoe = p->coefficient;
    tmpFre = p->frequency;

    //首项特殊处理输出，正号无需输出
    //0 次方不用输出 x，1 次方不用输出次方号
    if (tmpFre == 0) {
        printf("%d", tmpCoe);
    } else if (tmpFre == 1 && tmpCoe == 1) {
        printf("x");
    } else if (tmpFre == 1 && tmpCoe == -1) {
        printf("-x");
    } else if (tmpCoe == 1) {
        printf("x^%d", tmpFre);
    } else if (tmpCoe == -1) {
        printf("-x^%d", tmpFre);
    } else {
        printf("%dx^%d", tmpCoe, tmpFre);
    }

    p = p->next;
    while (p) {
        tmpCoe = p->coefficient;
        tmpFre = p->frequency;
```

```

        if (tmpFre == 1 && tmpCoe == 1) {
            printf("x");
        } else if (tmpFre == 1 && tmpCoe == -1) {
            printf("-x");
        } else if (tmpCoe == 1) {
            printf("+x^d", tmpFre);
        } else if (tmpCoe > 0) {
            printf("+%dx^d", tmpCoe, tmpFre);
        } else if (tmpCoe == -1) {
            printf("-x^d", tmpFre);
        } else {
            printf("%dx^d", tmpCoe, tmpFre);
        }
        p = p->next;
    }
    printf("\n");
}

```

//返回对 src 结点的一份深拷贝

```

Lnode *deepCpy(const Lnode *src) {
    Lnode *res;
    res = (Lnode *) malloc(sizeof(Lnode));
    res = (Lnode *) memcpy(res, src, sizeof(Lnode));

    return res;
}

```

3. （选做题）约瑟夫环



4-3.c

```

#include <stdio.h>
#include <stdlib.h>
#define Status int
#define OK 1
#define ERROR 0

typedef struct Lnode {
    int data;
    struct Lnode *next;
} Lnode, *Link;

Status createLoopLink (Link, int);

```

```

Status displayLink (Link);
Status solve (Link, int);

int main() {
    Link linklist;
    int n, k;

    linklist = (Link) malloc(sizeof(Lnode));
    linklist->data = -1;
    linklist->next = NULL;

    printf("input n: ");
    scanf("%d", &n);
    printf("input k: ");
    scanf("%d", &k);

    createLoopLink(linklist, n);
    //displayLink(linklist);

    solve(linklist, k);

    printf("\n");
    system("pause");
    return 0;
}

```

```

Status solve(Link head, int k) {
    int count=0;
    Link cur=head, pre=head;
    Lnode *tmp;
    //找到“尾结点”作为头结点的前一个结点
    while (pre->next != head)
        pre = pre->next;

    while (cur != cur->next) {
        count++;
        count %= k;
        //删除第 k 个结点
        if (count == 0) {
            printf("%d\n", cur->data);
            pre->next = cur->next;
            tmp = cur;
            cur = cur->next;
            free(tmp);
        }
    }
}

```

```

        } else {
            pre = cur;
            cur = cur->next;
        }
    }
    printf("%d", cur->data);
    free(cur);

    return OK;
}

Status createLoopLink (Link head, int n) {
    Lnode *tmp;
    Lnode *rear;
    head->data = 1;
    rear = head;
    for (int i = 2; i <= n; i++) {
        tmp = (Lnode *) malloc(sizeof(Lnode));
        if (tmp == NULL)
            return ERROR;
        tmp->data = i;
        tmp->next = NULL;
        rear->next = tmp;
        rear = tmp;
    }
    rear->next = head;

    return OK;
}

Status displayLink (Link head) {
    if (head == NULL)
        return ERROR;
    Link p=head;
    printf("%d", p->data);
    p = p->next;

    while(p) {
        printf("->%d", p->data);
        p = p->next;
    }
    return OK;
}

```

六、测试和结果

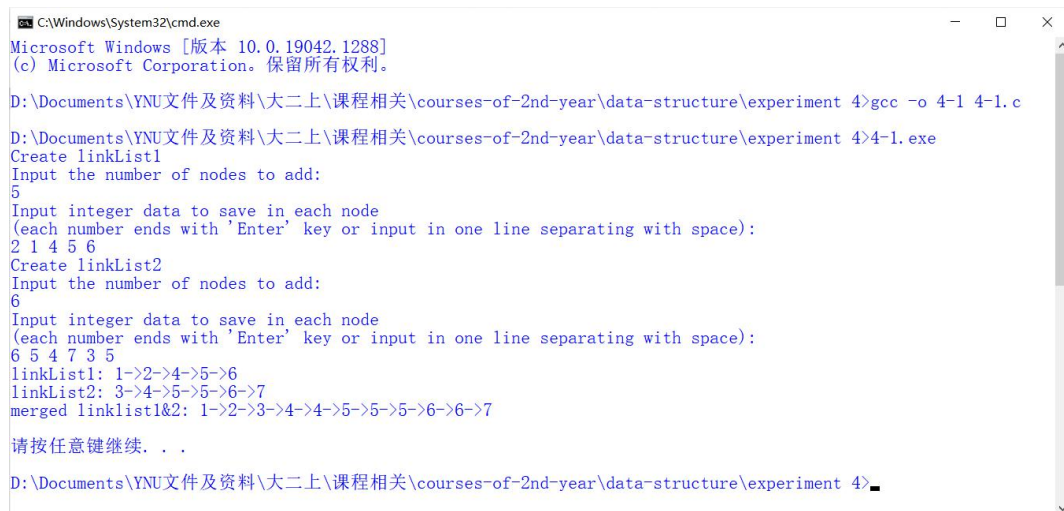
1. （必做题）有序线性表的构造与合并

Input:

```
5
2 1 4 5 6
6
6 5 4 7 3 5
```

Output:

```
linkList1: 1->2->4->5->6
linkList2: 3->4->5->5->6->7
merged linklist1&2: 1->2->3->4->4->5->5->5->6->6->7
```



```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19042.1288]
(c) Microsoft Corporation。保留所有权利。

D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 4>gcc -o 4-1 4-1.c
D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 4>4-1.exe
Create linkList1
Input the number of nodes to add:
5
Input integer data to save in each node
(each number ends with 'Enter' key or input in one line separating with space):
2 1 4 5 6
Create linkList2
Input the number of nodes to add:
6
Input integer data to save in each node
(each number ends with 'Enter' key or input in one line separating with space):
6 5 4 7 3 5
linkList1: 1->2->4->5->6
linkList2: 3->4->5->5->6->7
merged linklist1&2: 1->2->3->4->4->5->5->5->6->6->7

请按任意键继续. . .
D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 4>_
```

2. （必做题）多项式的构造与相加

Input:

```
createPoly(poly1, 2, 1, 2, 5, 6);
createPoly(poly2, 3, 1, 2, 1, 1, 1, 3);
```

Output:

```
x^2+5x^6
+x+x^2+x^3
=x+2x^2+x^3+5x^6
```

```
C:\Windows\System32\cmd.exe
D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 4>gcc -o 4-2 4-2.c
D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 4>4-2.exe
x^2+5x^6
+x+x^2+x^3
=x+2x^2+x^3+5x^6
请按任意键继续. . .
D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 4>
```

3. （选做题）约瑟夫环

Input:

10

4

Output:

4

8

2

7

3

10

9

1

6

5

```
C:\Windows\System32\cmd.exe
D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 4>gcc -o 4-3 4-3.c
D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 4>4-3.exe
input n: 10
input k: 4
4
8
2
7
3
10
9
1
6
5
请按任意键继续. . .
D:\Documents\YNU文件及资料\大二上\课程相关\courses-of-2nd-year\data-structure\experiment 4>
```

七、用户手册

1. （必做题）有序线性表的构造与合并

所有输入的数的范围都与 `int` 型一致。先输入要创建的第一个有序表中的元素个数，再依次输入各个元素数据域中 `data` 的值，以空格作为分隔；再输入要创建的第二个有序表中的元素个数，同样依次输入各个元素数据域中 `data` 的值，以空格作为分隔。

1. （必做题）多项式的构造与相加

此题为了函数的抽象通用性和低耦合性，将创建多项式的功能封装成了函数 `Status createPoly(Link head, int num, ...)`，即使用以 `head` 为头结点的链表存储，参数 `num` 为要创建的多项式的项数，后面的参数依次为对应的系数、次数的交替（应有 $2 \times \text{num}$ 个）。

如 `createPoly(poly1, 2, 1, 2, 5, 6)` 表示创建以 `poly1` 作为头结点存储的多项式，有 2 项，每一项的系数和指数分别为 1、2，5、6，即多项式 $x^2 + 5x^6$ ，需要注意的是，系数、指数的参数列表中不能有指数相同的两项。

2. （选做题）约瑟夫环

输入的 `n`、`k` 均与 `int` 型范围保持一致。