线性表

1.构造一个空的线性表

```
#include <iostream>
using namespace std;
typedef struct LNode *List;//定义线性表的结构体
struct LNode
{
    int Data;
    struct LNode *Next; //链表的尾指针指向下一节点，若无下一节点则指向NULL(所有尾指针都初始化为NULL)
};

List InitList(int n)
{
    List L, head, S;
    L = (List)malloc(sizeof(struct LNode)); //动态分配存储空间
    head = L;
    for (int i = 1; i <= n; i++)
    {
        int e;
        scanf("%d", &e);
        S = (List)malloc(sizeof(struct LNode)); //S是用来构造一个个节点的
        S->Data = e;
        S->Next = NULL; //尾指针初始化为NULL
        L->Next = S;    //L用来把一个个S节点串起来形成链表
        L = L->Next;    //指针后移
    }
    L = head->Next; //回到头节点
    free(head);     //释放head节点(head就是用来让L链表构造完成后回到头节点的)
    return L;
}
```

2.检测两个线性表的交集并输出(按第一个链表的顺序)

```
List Intersection(List L1, List L2)
{
    List L, t1, t2, s, head;
    t1 = L1, t2 = L2;
    L = (List)malloc(sizeof(struct LNode)); //L还是用来把一个个S节点串起来的(先初始化空链表)
    L->Next = NULL;
    head = L;
    while (t1)
    {
        t2 = L2;
        while (t2)
        {
            if (t2->Data == t1->Data)
            {
                s = (List)malloc(sizeof(struct LNode));
                s->Data = t1->Data; //S当作一个个节点被L串起来
                s->Next = NULL;
                L->Next = s;
                L = L->Next;
                break;
            }
            t2 = t2->Next;
        }
        t1 = t1->Next;
    }
    L = head->Next; //回到头节点
    free(head);
    return L;
}
```

3.输出一个线性表的值

```
void PrintList(List L)
{
    if (L == NULL)
        printf("\n");
    else
    {
        printf("%d", L->Data);
        L = L->Next;
        while (L)
        {
            printf(" %d", L->Data);
            L = L->Next;
        }
    }
}
```

4.循环链表

```
//经典约瑟夫环问题，用循环链表解决

#include <iostream>
using namespace std;
typedef struct CLinkList
{
    int data;
    struct CLinkList *next;
} node;
int main()
{
    //建立循环链表
    node *L, *r, *s;//L就是head，r就是L(本笔记1.部分的构造新线性表)，s就是S
    L = new node;//开辟新空间，建立新链表
    r = L;
    int n, m, i;
    cin >> n;//输入总人数
    cin >> m;//输入报哪个数的人出局
    int k = m;
    for (i = 1; i <= n; i++) //尾插法建立链表
    {
        s = new node;
        s->data = i;
        r->next = s;
        r = s;
    }
    r->next = L->next; //让最后一个结点指向第一个有数据结点(循环链表的关键一步，链表尾指针指向头部)
    node *p;
    p = L->next;
    delete L; //删除第一个空的结点
              //模拟解决约瑟夫问题
    while (p->next != p)//循环链表的退出条件
    {
        for (i = 1; i < k - 1; i++)
        {
            p = p->next;
        }
        cout << p->next->data << ' ';//输出出局的数
        p->next = p->next->next; //将该节点从链表上删除。
        p = p->next;//指针后移，指向下一个数
    }
    cout << p->data << endl;
    return 0;
}
```