数据结构 A
第二章参考答案

题 目 范 围 ：顺序表与单链表
授 课 老 师 ：何璐璐
助 教 ：王思怡（撰写人），赵守玺
联 系 邮 箱 ：2021302111095@whu. edu. cn

目录

1. 第一次作业

1.1. 单选题

题目1

在一个长度为n的顺序表中向第i个元素（1<=i && i<=(n+1)）之前插入一个新元素时需要向后移动 _____ 个元素。

A. n−i

B. n−i+1

C. n−i−1

D. i

答案：B

解析：在第i个元素之前插入一个新元素时，元素a_i~a_n都需要向后移动。

题目2

一个顺序表所占用存储空间的大小与 _____ 无关

A. 顺序表的长度

B. 顺序表中元素的数据类型

C. 顺序表中各数据项的数据类型

D. 顺序表中各元素的存放顺序

答案：D

解析：顺序表所占存储空间的大小=表长 * sizeof(元素的类型)，元素的类型显然会影响到存储空间的大小。对于同一类型的顺序表，表越长，则所占存储空间就越大。

题目3

顺序表具有随机存取特性指的是 _____

A. 查找值为x的元素与顺序表中元素的个数n无关

B. 查找值为x的元素与顺序表中元素的个数n有关

C. 查找序号为i的元素与顺序表中元素的个数n无关

D. 查找序号为i的元素与顺序表中元素的个数n有关

答案：C

解析：一种存储结构具有随机存取特性指的是，对于给定的序号i，在O(1)时间内找到对应元素值。

题目4

顺序表和链表相比存储密度较大，这是因为 _____

A. 顺序表的存储空间是预先分配的

B. 顺序表不需要增加指针来表示元素之间的逻辑关系

C. 链表的所有结点是连续的

D. 顺序表的存储空间是不连续的

答案：B

解析：顺序表不像链表要在结点中存放指针域，因此存储密度较大。

题目5

设某个线性表有n个元素，在以下运算中， _____ 在顺序表上实现比在链表上实现效率更高。

A. 输出第i（i<=n && i>=1）个元素的值

B. 顺序输出所有n个元素的值

C. 交换第1个元素与第2个元素的值

D. 求第1个值为x的元素的逻辑序号

答案：A

解析：A中，在顺序表中，可以直接通过基址加偏移的方式来访问第i个元素，其时间复

杂度为O(1)，即常数时间。在链表中，访问第i个元素需要从头节点开始，顺序遍历至第i个节点，其时间复杂度为O(i)，最坏情况下为O(n)。因此，顺序表在此操作上的效率显著高于链表。

B中，顺序表中的元素是连续存储的，可以直接顺序访问，总的时间复杂度为O(n)。而链表的元素是通过指针连接的，同样需要顺序遍历链表输出所有元素，时间复杂度也是O(n)。

C中，顺序表可以直接通过索引访问并交换第1个和第2个元素，操作的时间复杂度为O(1)。而链表同样可以直接访问并交换第1个和第2个元素的值，操作时间复杂度也为O(1)。

D中，顺序表需要从头至尾遍历顺序表直到找到值为x的元素，时间复杂度为O(n)。链表同样需要从头节点开始遍历直到找到值为x的节点，时间复杂度为O(n)。

题目6

以下关于单链表叙述正确的是 _____

ⅰ. 结点除自身信息以外还包括指针域，存储密度小于顺序表

ⅱ. 找第i个结点的时间为O(1)

ⅲ. 在插入、删除运算时不必移动结点

A. 仅i、iii

B. 仅i、ii

C. 仅ii、iii

D. 仅i、ii、iii

答案：A

解析：此题考察单链表的结构和操作特性。

ⅰ中，单链表的每个节点包含数据和至少一个指针域（通常是指向下一个节点的指针）。因为有额外的指针域，单链表的存储效率（存储密度）相较于顺序表（数组），其数据与总存储占用的比例更低。因此i正确。

ⅱ中，在单链表中访问第i个节点需要从头节点开始，逐个遍历直到第i个节点，因此

时间复杂度为O(i)，在最坏的情况下为O(n)，并非O(1)。因此，ii 错误。

iii中，在单链表中进行插入或删除操作时，只需修改相关节点的指针，不需要像顺序表那样移动其他元素来保持连续性。即，链表在插入和删除操作时不需要移动节点本身，只需处理指针。iii正确。

1.2. 编程题

1.2.1. 两数之和

【问题描述】

给定一个整数数组nums和一个目标值target，请你在该数组中找出和为目标值的那两个整数，并返回他们的数组下标。例如，给定nums=[2, 7, 11, 15]，target=9，返回结果为[0, 1]。若没有答案，则返回[-1, -1]。题目要求设计twoSum()函数：

```
class Solution {
public:
 vector<int> twoSum(vector<int>& nums, int target)
 { … }
};
```

【输入形式】

每个测试用例由两行数据构成，第一行给出整数数组，元素之间以空格隔开，可以有重复元素；第二行给出求和的目标值。

【输出形式】

返回累加和为目标值的两个整数在数组中的下标值，不限顺序。若没有答案，则返回[-1, -1]。

【样例输入】

```
2 7 11 15
9
```

【样例输出】

0 1

【样例说明】

输入的整数数组有四个整数，分别为2, 7, 11, 15，目标值为9。数组中的2和7的累加和为9，因此返回这两个整数在数组中的下标地址，即0, 1。测试数据文件名为in. txt。

参考答案：

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>

using namespace std;
class Solution {
public:
    vector<int> twoSum(vector<int>&nums,int target)
    {
        vector<int> ans;
        for(int i=0;i<nums.size()-1;i++)
            for (int j=i+1;j<nums.size();j++)
                if (nums[i]+nums[j]==target)
                {
                    ans.push_back(i);
                    ans.push_back(j);
                    return ans;
                }

        ans.push_back(-1);
        ans.push_back(-1);
        return ans;  // no answer, return -1 -1
    }
};
int main() {
    ifstream inFile;
    inFile.open("in.txt", ios::in);
    if(!inFile){
        cout << "error open in.txt!" << endl;
        return 1;
    }
    string s;
    getline(inFile, s);    // read a line
    int x;
    vector<int> nums;
```

```
37.     int target;
38.     istringstream strin(s);
39.     while (strin >> x){
40.         nums.push_back(x);
41.     }
42.     inFile >> target;
43.
44.     Solution obj;
45.     vector<int> ans;
46.     ans = obj.twoSum(nums, target);
47.     cout << ans[0] << ' ' << ans[1];
48.
49.     inFile.close();
50.     return 0;
51.}
```

解析：

这道题目是一个经典的“两数之和”问题，主要由以下三部分组成：

- 嵌套循环：通过双重循环遍历数组，检查每一对元素的和是否等于目标值 target。
- 及时返回：一旦找到符合条件的一对元素，立即返回它们的索引。
- 无结果处理：如果遍历完数组后没有找到符合条件的元素对，则返回 [-1, -1]。

以下是同学们解题时需要注意的几点：首先，该方法虽然可以解决问题，但性能一般——时间复杂度是 O(n^2)，因为每个元素都被与其他元素比较一次；其次，要注意边界条件，代码中未显示处理输入数组为空的情况，实际应用中需要注意这一点；最后，要熟练文件操作，解题时需要我们确保文件格式正确且可读。

感兴趣的同学可以思考一下代码如何优化，所谓“技多不压身”，比如使用哈希表（或称为字典）来减少查找时间。思路是在遍历数组的同时，用一个哈希表记录已经遍历过的数字及其索引，以便快速检查每个数字的配对数字（即 target - 当前数字）是否已存在。

1.2.2. 重排链表

【问题描述】

假设不带头结点的单链表结点类型如下：

```
struct ListNode
{ int val;
 ListNode *next;
 ListNode(int x):val(x),next(NULL){}
};
```

给定一个含n+1个结点的单链表L：L0→L1→…→Ln−1→Ln，将其重新排列后变为：L0→Ln→L1→Ln−1→L2→Ln−2→…。你不能只是单纯的改变结点内部的值，而是需要实际的进行结点交换。示例1，给定链表为1−>2−>3−>4，重新排列为1−>4−>2−>3。示例2，给定链表为1−>2−>3−>4−>5，重新排列为1−>5−>2−>4−>3。你需要设计的reorderList函数。

```
class Solution {
public:
 void reorderList(ListNode * head)
 { … }
};
```

【输入形式】

输入含有n个元素的单链表。

【输出形式】

输出重排后的单链表中的元素。

【样例输入】

1 2 3 4

【样例输出】

1 4 2 3

【样例说明】

输入文件名为in. txt。

参考答案：

```
#include <iostream>
#include <fstream>
#include <vector>
```

```
4. using namespace std;
5. struct ListNode
6. {    int val;
7.      ListNode *next;
8.      ListNode(int x):val(x),next(NULL){}
9. };
10.
11.class Solution
12.{ ListNode *Findmiddle(ListNode *head)    //查找中间位置的结点
13.     { ListNode *slow=head,*fast=head;
14.         while (fast!=NULL && slow!=NULL)
15.         { if (fast->next==NULL || fast->next->next==NULL)
16.                 return slow;        //满足结束条件时返回
17.             slow=slow->next;       //慢指针每次后移 1 个结点
18.             fast=fast->next->next;       //快指针每次后移 2 个结点
19.         }
20.         return slow;
21.     }
22.     void Reverse(ListNode *&head1)     //逆置不带头结点的单链表 head1
23.     { ListNode *p=head1,*q;
24.         head1=NULL;         //初始置为空
25.         while (p!=NULL)        //遍历所有结点 p
26.         { q=p->next;
27.             p->next=head1;        //将结点 p 插入到表头
28.             head1=p;
29.             p=q;
30.         }
31.     }
32.     void Union(ListNode *&head,ListNode *head1) //合并 head 和 head1 的结点
33.     { ListNode *p=head->next,*q=head1,*r;
34.         r=head;           //r 为尾结点指针
35.         while (p!=NULL || q!=NULL)      //遍历所有结点
36.         { if (q!=NULL)
37.             { r->next=q; r=q;       //结点 q 插入到 head 的表尾
38.                 q=q->next;
39.             }
40.             if (p!=NULL)
41.             { r->next=p; r=p;       //结点 p 插入到 head 的表尾
42.                 p=p->next;
43.             }
44.         }
45.         r->next=NULL;         //尾结点 next 置为空
46.     }
47.public:
48.     void reorderList(ListNode* head)     //求解算法
49.     { if (head==NULL || head->next==NULL)
```

```
            return;
        ListNode *head1,*p;
        p=Findmiddle(head);
        head1=p->next;
        p->next=NULL;          //head 断开为 head+head1
        Reverse(head1);          //逆置 head
        Union(head,head1);          //head 和 head1 合并为 head
    }

    ListNode* create(vector<int>&nums){     // 尾插法创建单链表
        ListNode *s, *r, *head;
        if(!nums.empty()){
            s = new ListNode(nums[0]);
            r = s;
            head = s;
            for(int i = 1; i < nums.size(); i++){
                s = new ListNode(nums[i]);
                r->next = s;
                r = s;
            }
            r->next = NULL;
        }
        else head = NULL;
        return head;
    }

    void printList(ListNode *head) {
        ListNode *p;
        p = head;
        while (p) {
            cout << p->val << ' ';
            p = p->next;
        }
        cout << endl;
    }
};
int main() {
    ifstream inFile;
    inFile.open("in.txt", ios::in);
    if(!inFile){
        cout << "error open in.txt!" << endl;
        return 1;
    }

    vector<int> nums;
    while(!inFile.eof()){
```

```
96.         int x;
97.         inFile >> x;
98.         nums.push_back(x);
99.     }
100.        Solution obj;
101.        ListNode *p;
102.        p = obj.create(nums);
103.        obj.reorderList(p);
104.        obj.printList(p);
105.        inFile.close();
106.        return 0;
107.    }
```

解析：

此题的解题思路主要时以下三个步骤：

- 寻找链表中点： 使用快慢指针法寻找链表的中点。快指针每次移动两步，慢指针每次移动一步，当快指针到达链表末尾时，慢指针则位于链表中点。这一步骤是分割链表的前提，确保能正确将链表分为两部分。
- 链表后半部分逆序： 将中点之后的链表部分逆序。逆序后，我们可以从链表的两端开始交替合并。逆序是通过不断地取出元素，并将其插入到新链表的头部实现的。
- 链表合并： 将前半部分和逆序后的后半部分交替合并。这一步是实现最终目标的关键，通过交替合并，达到题目要求的链表重排。

使用快慢指针寻找中点是链表操作中的常用技巧，可以有效地找到中点而不需要知道链表的具体长度。逆序链表的操作是因为重排后的顺序需要从两头向中间排列，逆序可以方便地实现从后向前的访问和合并。

想额外提醒同学们注意的是，编写程序时应当主动保持代码的清晰和模块化，养成良好的编程习惯，便于开发。此题中如创建链表、逆序、合并和打印链表都应该是独立的函数。

2. 第二次作业

2.1. 顺序表元素划分

【问题描述】

若一个线性表采用顺序表L存储，其中所有元素为整数。

设计一个时间空间两方面尽可能高效的算法，将所有元素划分为三部分，三部分以K1和K2为界。

划分举例：

对于（6，4，10，7，9，2，10，1，3，30），已知K1=5，K2=8，一种划分结果为（3，4，1，2，6，7，20，10，9，30）

如果K1<=K2, 输出一种划分；否则输出false。

【输入形式】

第一行输入两个数K1和K2。

第二行输入线性表的数据个数N（N<30）。

第三行输入线性表的整数数据。

【输出形式】

如果K1<=K2, 输出一种划分；否则输出false。

【样例输入】

5 8

10

6 4 10 7 9 2 10 1 3 30

【样例输出】

3 4 1 2 6 7 20 10 9 30

参考答案：

```
#include <iostream>
using namespace std;
```

```
void move(int* L, int k1, int k2, int n )
{
    int i = -1, j = 0, k = n;
    while (j < k)
    {
        if (L[j] < k1)
        {
            i++;
            swap(L[i], L[j]);
            j++;
        }
        else if (L[j] > k2)
        {
            k--;
            swap(L[j], L[k]);
        }
        else
        {
            j++;
        }
    }

}

int main()
{
    int k1, k2, n;
    int elem, i = 0;
    cin >> k1 >> k2 >> n;
    int a[30];
    while (cin >> elem)
    {
        a[i] = elem;
        i++;
    }
    move(a, k1, k2, n);
    for (int index = 0; index < n; index++)
    {
        cout << a[index] << " ";
    }
    return 0;
}
```

解析：

这个顺序表元素划分问题，我们主要运用的方法是三向分区，这是一种高效处理数

组分类问题的算法思路，通常用于解决涉及范围划分的场景。同学们可以参考“线性表小节. ppt”中的荷兰旗问题，本质思路是一致的。

三向分区是快速排序中用来应对包含大量重复元素的优化方案，但其本质是将数组分为三个部分：小于某个值、等于某个值和大于某个值。在本题中，这个方法被用来将数组分为小于K1、介于K1和K2之间、以及大于K2三部分。这种方法可以有效地在一次遍历中完成分区，提高算法的时间效率。

方法涉及三个指针：两个用于定义当前已知的分区边界，一个用于探索未知区域。在单次遍历中按照指定的界限K1和K2调整元素位置，达到分区目的。

2.2. 集合并集运算

【问题描述】

给你两个集合A和B，要求{A}+{B}。注意同一个集合中不会有两个相同的元素。

【输入形式】

每组输入数据分为三行，第一行有两个数字n，m（0<n，m≤10000），分别表示集合A和集合B的元素个数。后两行分别表示集合A和集合B，每个元素为不超出int范围的整数，两个元素之间有一个空格隔开。

【输出形式】

针对每组数据输出一行数据，表示合并后的集合，要求从小到大输出，两个元素之间有一个空格隔开。

【样例输入】

1 2

1

2 3

【样例输出】

1 2 3

【样例说明】

第一行的两个数字表示集合A和集合B的元素个数，后面两行分别表示集合A和集合

B中的整数元素，两个元素之间用空格隔开。测试数据存放在in. txt文件中。

参考答案：

```
#include <iostream>
#include <fstream>
#include <list>
using namespace std;
void solve(list<int> &A,list<int> &B,list<int> &C) //二路归并得到C
{
    list<int>::iterator ita=A.begin();
    list<int>::iterator itb=B.begin();
    while (ita!=A.end() && itb!=B.end())    //A,B 均没有遍历完
    { if (*ita<*itb)          //将较小的A 中元素添加到C 中
        { C.push_back(*ita);
            ita++;
        }
        else if (*ita>*itb)         //将较小的B 中元素添加到C 中
        { C.push_back(*itb);
            itb++;
        }
        else            //相等时只添加一个到C 中
        { C.push_back(*ita);
            ita++; itb++;
        }
    }
    while (ita!=A.end())        //A 没有遍历完，剩余元素添加到C 中
    { C.push_back(*ita);
        ita++;
    }
    while (itb!=B.end())        //B 没有遍历完，剩余元素添加到C 中
    { C.push_back(*itb);
        itb++;
    }
}
int main()
{
    ifstream inFile;
    inFile.open("in.txt", ios::in);
    if(!inFile){
        cout << "error open in.txt!" << endl;
        return 1;
    }
    int n,m,x;
    if (!inFile.eof()){
        inFile >> n >> m;
```

```
43.         list<int> A,B,C;
44.         for (int i=0;i<n;i++)
45.         {
46.             inFile >> x;
47.             A.push_back(x);
48.         }
49.         for (int i=0;i<m;i++)
50.         {
51.             inFile >> x;
52.             B.push_back(x);
53.         }
54.         A.sort();           //A 中元素递增排序
55.         B.sort();           //B 中元素递增排序
56.         solve(A,B,C);         //A,B 归并得到C
57.         list<int>::iterator itc=C.begin();
58.         bool first=true;
59.         while (itc!=C.end())        //输出C 中元素
60.         {
61.             if (first)
62.             {
63.                 cout << *itc;
64.                 first=false;
65.             }
66.             else cout << " " << *itc;
67.             itc++;
68.         }
69.         cout << endl;
70.     }
71.     inFile.close();
72.     return 0;
73.}
```

解析：

这个问题的目标是将两个集合A和B合并成一个新的集合C，并且输出时集合C中的元素按升序排列，同时排除任何重复元素。这是一个经典的集合操作问题，涉及到元素合并、排序、和去重。

解题首先想到的是分别对集合A和B进行排序，这是为了利用两路归并的技术来有效地合并两个有序集合。在归并两个集合的时候，如果集合已经是有序的，那么归并操作会更加高效，因为我们可以直接比较两个列表的头部元素来决定哪个元素应该先被添加到结果集合中。

归并使用两个迭代器分别遍历集合A和B，根据大小关系将元素添加到集合C中。如

果两个集合中的元素相等，则只添加其中一个到集合C中，实现去重。完成A和B的初始归并后，将未处理完的集合中的剩余元素添加到集合C中。

同学们应该注意在读取文件时要检查文件是否成功打开，防止程序在运行时因文件不存在或路径错误而崩溃，也能帮助自己debug。

2.3. 归并排序

【问题描述】

有一个含n（n<=200000）个整数的无序序列，采用链表的二路归并排序实现递增排序

【输入形式】

一行字符串，包含多个整数，每个数之间用空格分开。

【输出形式】

递增排序的结果，每个数之间用空格分开。

【样例输入】

9 4 7 6 2 5 8 1 3

【样例输出】

1 2 3 4 5 6 7 8 9

【样例说明】

测试数据的文件名为in. txt，输出文件名为out. txt

参考答案：

```
#include <iostream>
#include <string>
#include <fstream>
#include <stdio.h>
#include <vector>

using namespace std;
struct ListNode{
 int val;
ListNode* next;
ListNode() : val(0), next(nullptr) {}
ListNode(int x) : val(x), next(nullptr) {}
ListNode(int x, ListNode* next) : val(x), next(next) {}
```

```
};

ListNode* merge(ListNode* head1, ListNode* head2)
{
    ListNode guard_node(0);
    ListNode* mList = &guard_node;
    while (head1 && head2)
    {
        if (head1->val <= head2->val)
        {
            mList->next = head1;
            head1 = head1->next;
        }
        else
        {
            mList->next = head2;
            head2 = head2->next;
        }
        mList = mList->next;
    }
    if (head1)
        mList->next = head1;
    if (head2)
        mList->next = head2;
    return guard_node.next;
}

ListNode* sortList(ListNode* head)
{
    if (head == NULL || head->next == NULL) return head;
    ListNode* fast = head;
    ListNode* slow = head;
    while (fast->next && fast->next->next)
    {
        slow = slow->next;
        fast = fast->next->next;
    }
    ListNode* head2 = slow->next;
    slow->next = NULL;
    ListNode* head1 = head;
    head1 = sortList(head);
    head2 = sortList(head2);
    return merge(head1, head2);
}

int main()
```

```
{
 freopen("in.txt", "r", stdin);
    ListNode* head = new ListNode();
    ListNode* rear = head;
 int num;
 while (cin >> num)
 {
        ListNode* t = new ListNode(num);
        rear->next = t;
        rear = t;

 }
 freopen("out.txt", "w", stdout);

    ListNode* i = sortList(head->next);
    if (i != nullptr)
    {
        while (i->next != nullptr)
        {
            cout << i->val << ' ';
            i = i->next;
        }
        cout << i->val << endl;
    }

 return 0;

}
```

解析：

解题思路使用了递归的思想，每次将链表从中点断开，形成两个独立的链表，并递归地对这两个链表进行排序。

sortList函数递归地将链表分成两半，直到每个子链表只有一个节点或为空。使用快慢指针来找到链表中点：快指针每次移动两步，慢指针每次移动一步，当快指针到达链表尾部时，慢指针将位于链表中点。merge函数用于合并两个已排序的链表。通过创建一个哨兵节点（guard_node）来简化合并操作，避免处理空链表的特殊情况。

2.4. 好玩的约瑟夫环-单链表版本

【题目描述】

有M个人，编号分别为1到M，玩约瑟夫环游戏，最初时按编号顺序排成队列；每遍游戏开始时，有一个正整数报数密码N，队列中人依次围坐成一圈，从队首的人开始报数，报到N的人出列，然后再从出列的下一人开始重新报数，报到N的人出列；重复这一过程，直至所有人出列，完成一遍游戏，所有出列的人形成新队列；游戏可能玩很多遍，每遍有新报数密码。求若干遍游戏完成后队列次序。本题要求使用单链表实现，程序要求采用模块化设计，格式规范，有合适注解。

【输入描述】

每个测试用例包含若干个正整数（至少1个），第一个正整数为玩游戏人数M，后续每个正整数为每遍游戏报数密码；报数密码可能为1

【输出描述】

每个测试用例结果占一行，每个编号占4位。

【样例输入】

10 3 5 2

【样例输出】

4 6 5 2 9 1 3 7 8 10

参考答案：

```
#include <iostream>
using   namespace   std;
class   CJosephRing {
    struct Node {
        int     data;
        Node    *next;
    }  *m_pLast;
public :
    CJosephRing ();
    void  Display() const ;
    ~CJosephRing () {
        _Clear ();
    }
    void  Append (int x);
    void  RunGame (int N, CJosephRing &result);
private :
    void    _Clear ();
};
```

```
CJosephRing::CJosephRing ()
{
   m_pLast = NULL;
}
void  CJosephRing::Display() const
{
    if (m_pLast == NULL)
        return;
    for (Node *p = m_pLast->next; ;p = p->next) {
        cout.width (4);
        cout << p->data;
        if (p == m_pLast)
            break;
    }
    cout << endl;
}

void    CJosephRing::_Clear ()
{
    if (m_pLast == NULL)
        return;
    while (m_pLast->next != m_pLast) {
        Node *p = m_pLast->next;
        m_pLast ->next = p->next;
        delete p;
    }
    delete m_pLast;
}

void  CJosephRing::Append (int x)
{
    Node *p = new Node;
    p->data = x;
    if (m_pLast == NULL) {
        p->next = p;
        m_pLast = p;
        return;
    }
    p->next = m_pLast->next;
    m_pLast ->next = p;
    m_pLast = p;
}

void  CJosephRing::RunGame (int N, CJosephRing &result)
{
    if (m_pLast == NULL)
```

```
        return;
    while (m_pLast->next != m_pLast) {
        for (int i = 1; i < N; i++)
            m_pLast = m_pLast->next;
        Node *p = m_pLast->next;
        m_pLast ->next = p->next;
        result.Append(p->data);
        delete p;
    }
    result.Append(m_pLast->data);
    delete m_pLast;
    m_pLast = NULL;
}

int main()
{
    int     M,N;
    int     times =  0;

    cin>>M;
    CJosephRing     firstRing, secondRing;

    for (int i = 1; i <= M; i++) {
        firstRing.Append(i);
    }
    while  (cin >>N) {
        if (times % 2 == 0)
            firstRing.RunGame(N, secondRing);
        else
            secondRing.RunGame(N, firstRing);
        ++times;
    }
    if (times % 2 == 0)
        firstRing.Display();
    else
        secondRing.Display();
}
```

解析：

我们首先建立一个环形链表来模拟人员的排列，然后通过模拟每轮游戏进行报数以达到出列的目标。重点是在于链表的操作，包括节点的插入、删除以及整个链表的清理。

以下是解题思路：

1. 链表节点定义

在这个问题中，我们使用单链表的节点定义，每个节点包含两个部分：存储数据的data字段，以及指向下一个节点的指针next。为了方便操作链表的尾部，我们还保持一个指针m_pLast指向链表的最后一个节点。

2. 环形链表的构建

初始时，根据输入的人数M，我们构建一个大小为M的环形链表。每个人的编号从1到M依次进入链表。构建过程中需要特别注意节点的插入，以确保链表在插入完所有节点后成环。

3. 约瑟夫环的游戏逻辑

游戏的每一轮从报数1开始，数到N的人将出列，并从下一个人重新开始数数。为了实现这一逻辑，我们需要：从链表头部开始，顺时针移动N−1步到达第N个人；将这个人从链表中移除，并将其添加到结果链表中；继续从下一个节点开始新一轮的数数。

4. 游戏的多轮进行

由于题目中可以进行多轮游戏，每一轮使用新的报数密码N，我们需要处理多次输入的N。为了模拟这种多次操作，我们可以交替使用两个链表对象，一个作为当前轮的输入链表，另一个作为输出链表，每轮结束后交换二者角色继续下一轮。