组号: 5

上海大学计算机工程与科学学院

实 验 报 告

（数据结构 1）

学　　期：2024-2025 年冬季

组　　长：冯俊佳

学　　号：23122721

指导教师：朱能军

成绩评定：　　（教师填写）

二〇二四年十一月二十九日

小组信息				
登记序号	姓名	学号	贡献比	签名
1	冯俊佳	23122721	1/3	冯俊佳
2	钱傲栋	23122803	1/3	钱傲栋
3	邱添	23122720	1/3	邱添

实验概述	
实验零	（熟悉上机环境、进度安排、评分制度；确定小组成员）
实验一	抽取简历
实验二	
实验三	
实验四	

实验一

1 实验题目

1.1 抽取简历（1）

1.1.1 问题描述

某 IT 公司招聘新员工，已收到 N 份简历，人力资源部 X 和 Y 负责挑选简历安排面试。他们把 N 份简历排成一个圆圈，按逆时针方向编号为 1～N。开始时 X 站在 1 号简历前，按逆时针方向数到第 K 份简历，选中；Y 站在 N 号简历前，按顺时针方向数到第 M 份简历，选中。两人同时取走所选简历后，分别按逆时针和顺时针走到下一份简历前，然后 X 和 Y 再重复上述方法取简历，直到取走全部简历，如果两人选中同一份简历，则只输出一个编号。

1.1.2 基本要求

要求输入 3 个数 N、K 和 M，按取走简历的顺序（先甲后已）输出简历编号。

1.1.3 测试数据

输入样例：

10 4 3

输出样例：

4, 8; 9, 5; 3, 1; 2, 6; 10; 7

1.2 抽取简历（2）

1.2.1 问题描述

某 IT 公司招聘新员工，已收到 N 份简历，人力资源部 X 和 Y 负责挑选简历安排面试，Z 负责补充新的简历。他们把 N 份简历排成一个圆圈，按逆时针方向编号为 1～N。开始时 X 站在 1 号简历前，按逆时针方向数到第 K 份简历，选中；Y 站在 N 号简历前，按顺时针方向数到第 M 份简历，选中。两人同时取走所选简历后，分别按逆时针和顺时针走到下一份简历前，此时（除非所有简历都已取完或者刚刚 X 和 Y 取走的是同一份简历），Z 会拿来 1 份新简历（编号从 N+1 开始

递加），插到 X 前面，然后 X 和 Y 再重复上述方法取简历，直到取走全部简历，如果两人选中同一份简历，则只输出一个编号。

1.2.2 基本要求

要求输入 3 个数 N、K 和 M，按取走简历的顺序（先甲后已）输出简历编

1.2.3 测试数据

输入样例：

5 1 1

输出样例：

1, 5; 6, 4; 7, 3; 8, 2

2 实验内容

利用数组或双向循环链表来实现以上问题，以达到熟练使用数组和双向循环链表数据结构的目的，总结规律，深刻理解数组和双向循环链表的特点。

3 解决方案

3.1 算法设计

3.1.1 数据结构

本实验要实现的目的是结点的循环，因此首先考虑的是使用数组或者顺序。由于第二问需要实现对该循环的增减，而数组缺少对成员增删的功能，因此考虑使用具有此功能的数据结构——链表。

在对链表的选择中，有带头结点的双向链表和不带头结点的双向链表两种选择，由于题目要求的是实现循环的功能，而如果选择带头结点的话在寻找结点前驱时的处理比较复杂，因此本实验采取不带头结点的双向循环链表作为数据结构。

此外，本实验本着研讨钻研的理念，对第一问也设计了用顺序表来实现。（在 3.4 中会单独加以说明，后续内容仅讨论用链表实现）

3.1.2 算法思想

本代码的思想为：在初始化时，设置两根指针指向链表的头结点和尾部结点

（即头结点的前驱），分别代表题目中 x 与 y 的初始站位。之后将指针分别顺时针与逆时针移动 k 与 m 个结点，此时需要做判断：如果两根指针指向同一个结点，即题目中的 x 和 y 走到了同一份简历的面前，输出当前简历的序号并删除该简历；若指向不同的结点，则分别输出指向的结点的序号并删除这两个结点。

3.1.3 主要操作

在代码中，主要需要实现的功能有：新增结点（在传入 1~N 的简历中使用），删除结点（在 x 与 y 分别拿走简历时使用），插入结点（在第二问中新放入的简历 Z）。

3.2 源程序代码

以下简介几个该程序的核心代码，其中包括：新增结点、删除结点以及主函数等。

3.2.1 尾插法新增结点

TailInsert 函数实现的是传入数据、新增结点，是对于链表的尾插功能，具体应用在初始化创建链表的时候，如下所示。

```
1.     void DblLinkList::TailInsert(int d)
2.     {
3.         Node *p;
4.         if(head==NULL)  //如果头指针为空，代表链表为空，在头指针的位置新增结点
5.         {
6.             head=new Node(d);
7.             head->next=head;
8.             head->prior=head;
9.             length++;
10.            return;
11.        }
12.        p=new Node(d);       //否则在尾部新增一个结点
13.        length++;
14.        head->prior->next=p;
15.        p->prior=head->prior;
16.        head->prior=p;
17.        p->next=head;
18.    }
19.
```

```
20.     cin >> n;          //输入简历的份效
21.     for(int i=1;i<=n;i++) DLL.TailInsert(i); //用简历的编码创建链表
```

3.2.2 在某个位置新增结点

Insert 函数实现的是传入数据以及链表中的一个结点，并在该结点之前新增结点的插入功能，具体应用在问题 2 需要在 x 前插入新增简历（结点）时使用。具体代码如下所示。

```
1.      void DblLinkList::Insert(int d,Node *p)
2.      {
3.          Node *q;
4.          q=new Node(d,p->prior);       //创建新结点，改变指针指向使新结点增加到传入结点的前面
5.          p->prior->next=q;
6.          q->next=p;
7.          p->prior=q;
8.          length++;                     //链表长度+1
9.      }
```

3.2.3 删除结点

Delete 函数实现的是传入链表中的一个结点并删除该结点，同时返回该结点的数据域的功能，在取走简历（消除结点）时使用。具体代码如下所示。

```
1.      int DblLinkList::Delete(Node *p)  //传入结点 p，删除该结点
2.      {
3.          int d=p->data;      //将 d 作为返回值，记录 p 结点的数据域
4.          p->prior->next=p->next;
5.          p->next->prior=p->prior;
6.          delete p;
7.          length--;
8.          return d;
9.      }
```

3.2.4 主函数

主函数分为初始化部分和运行部分。

初始化部分如下所示。

```
1.      DblLinkList DLL;
2.      cin >> n;          //输入简历的份效
3.      for(int i=1;i<=n;i++) DLL.TailInsert(i);//用简历的编码创建链表
4.      Node *x=DLL.head;             //x 指向链表的头指针，即第一份简历 1
```

```
5.      Node *y=DLL.head->prior;  //y 指向链表的尾指针，即最后一份简历 N
```

运行部分如下图代码所示。

```
1.      while(DLL.IsEmpty()==false)   //特判:如果链表非空，则继续循环
2.      {
3.          for(int i=1;i<k;i++) x=x->next;  //数了 k 份简历，即移动 k 个结点，改变 x 的指向
4.          q1=x;
5.          x=x->next;
6.          for(int i=1;i<m;i++) y=y->prior; //数了 m 份简历，即移动 m 个结点，改变 y 的指向
7.          q2=y;
8.          y=y->prior;
```

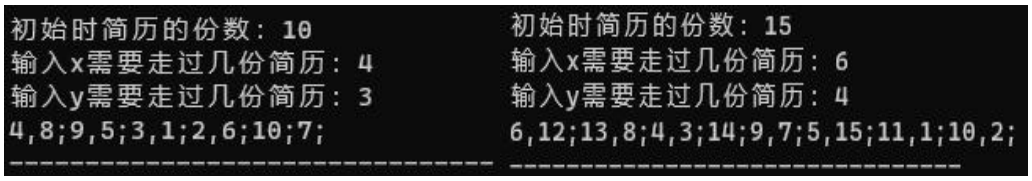
由于需要删除结点同时保留 x 与 y 的指向，因此本代码使用了两根指针 q1 和 q2 来指向需要删除的结点。如果两根指针所指向的结点为同一个结点，那么只需删除该结点并且返回该结点的数据；如果分别指向两个结点，先用 q1 和 q2 暂存两个结点，再将 x 和 y 分别顺时针、逆时针移动一位，最后做 q1 和 q2 指向的结点的删除。需要注意的是，有以下一种特殊情况：如果要 x 拿走的简历恰好是 y 移动后面对的简历，那其实 y 所面对的没有简历，即 y 应该再逆时针走一步。用代码理解的话，就是 y 所指向的结点为空，因此 y 移动到它的前驱结点。X 指针同理。代码如下所示。

```
1.      while(DLL.IsEmpty()==false)   //特判:如果链表非空，则继续循环
2.      {
3.          for(int i=1;i<k;i++) x=x->next;  //数了 k 份简历，即移动 k 个结点，改变 x 的指向
4.          q1=x;
5.          x=x->next;
6.          for(int i=1;i<m;i++) y=y->prior; //数了 m 份简历，即移动 m 个结点，改变 y 的指向
7.          q2=y;
8.          y=y->prior;
9.
10.         if(q1==q2) cout << DLL.Delete(q1) << " "; //如果指向相同，删除该结点
11.         else
12.         {
13.             if(y==q1) y=y->prior;
14.             if(x==q2) x=x->next;
```

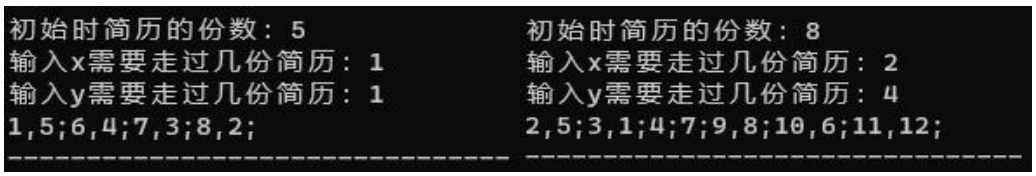
```
15.             cout << DLL.Delete(q1) << " " << DLL.Delete(q2) << " ";

16.
17.             if (DLL.IsEmpty()) return 0;
18.             DLL.Insert(++n,x);              //在 x 前新增结点
19.             x=x->prior;                     //x 位置移动
20.         }
21.     }
22.     return 0;
```

3.3 运行结果

3.3.1 面试安排 1 的运行结果：



3.3.2 面试安排 2 的运行结果：



经过验算，本程序可以实现预期的运行结果。

3.4 用顺序表实现

在本实验中，我们也尝试设计用顺序表实现相关问题，解决了第一问。但是经过研究后发现用顺序表实现的效果并不理想，具体原因如下：

•顺序表删除元素时会导致顺序表的长度和下标元素发生改变。在本题中体现在抽取简历时有先后顺序，在抽取序号较小的简历后，会对抽取另一份简历产生影响。

•在代码实现时，需要通过取模运算来实现顺序表的循环，即当下标超出顺序表的容量范围时，需要用取模运算使其回到顺序表下一次取简历的正确位置上。而取模运算需要使用到顺序表当前的长度，而先前删除元素的同时又会改变顺序表的长度。因此，使得顺序表的循环容易出错。

•顺序表的优势未能体现。顺序表在随机读取元素时的时间复杂度为 O（1），但是在本题中几乎未使用到随机读取，因此没能凸显出顺序表相比于链表的优势。

具体核心代码如下：

```
// 简历选择逻辑
template <class ElemType>
void SeqList<ElemType>::Select(int n, int k, int m) {
    int cur_IndexX = 0;         // X的起始位置(数组索引从0开始)
    int cur_IndexY = n - 1;     // Y的起始位置

    while (length > 0) {
        // 计算X的选择位置（逆时针走K步）
        if (cur_IndexY < 0){
            cur_IndexY += length;
        }
        cur_IndexX = (cur_IndexX + k - 1) % length;
        ElemType x_selected = elems[cur_IndexX];

        // 计算Y的选择位置（顺时针走M步）
        cur_IndexY = (cur_IndexY - m + 1 + length) % length;
        ElemType y_selected = elems[cur_IndexY];

        // 输出X和Y的选择结果
        if (cur_IndexX == cur_IndexY) {
            // 如果X和Y选择的是同一份简历
            cout << x_selected;
            DeleteElem(cur_IndexX + 1, x_selected);  //删除该简历
        }
        else {
            // 如果X和Y选择不同简历
            cout << x_selected << ", " << y_selected;
            if (cur_IndexX < cur_IndexY) {
                DeleteElem(cur_IndexY + 1, y_selected); //先删除Y的位置
                DeleteElem(cur_IndexX + 1, x_selected); //再删除X的位置
            }
            else {
                DeleteElem(cur_IndexX + 1, x_selected); //先删除X的位置
                DeleteElem(cur_IndexY + 1, y_selected); //再删除Y的位置
            }
        }
```

```
37.
38.             if (length > 0) {
39.                 cout << "; ";  // 输出分隔符
40.             }
41.
42.     // 删除后需要调整 X 和 Y 的索引位置，对应题目中的“走到下一份简历前”
43.             // 若 X 的当前索引等于 Y 的当前索引，则将 Y 的索引往前一位。
44.             if (cur_IndexX == cur_IndexY) {
45.                 cur_IndexY--;
46.                 goto flag;
47.             }
48.             // 若 X 的当前索引小于 Y 的当前索引，则将 Y 的索引往前两位。
49.             if (cur_IndexX < cur_IndexY) {
50.                 cur_IndexY-=2;
51.                 goto flag;
52.             }
53.           // 若X的当前索引大于Y的当前索引,则将X和Y的索引各往前一位。
54.             if (cur_IndexX > cur_IndexY) {
55.                 cur_IndexX--;
56.                 cur_IndexY--;
57.                 goto flag;
58.             }
59.         flag:;
60.         }
61.         cout << endl;
62.     }
```

4 算法分析

4.1 算法时间复杂度分析

主循环中的操作：主函数的循环体首先执行若干次 x=x->next 和 y=y->prior，这些操作每次的时间复杂度是 O(1)；然后，检查 q1 和 q2 是否相等，如果相等调用 Delete(q1)，否则调用 Delete(q1)和 Delete(q2)两次，每次删除的时间复杂度是 O(n)（因为 Delete(int n)最坏情况下需要遍历 n 个节点）；如果 q1!= q2，还需要调用 Insert(++n, x)插入新节点，时间复杂度是 O(1)。

因此，总时间复杂度是：最坏情况下，每次删除操作的复杂度为 O(n)，循环最多执行 n 次，因此总时间复杂度是 $O(n^2)$。

4.2 算法空间复杂度的分析

每个 Node 结点包含了三个字段：prior、next 和 data，每个节点占用 O(1)的空间；DblLinkList 类的空间消耗：head 指针占用 O(1)的空间，length 变量占用 O(1)的空间；主程序中的空间消耗：在主函数中，创建了一个 DblLinkList 对象，空间消耗为 O(n)，其中 n 是链表的长度，同时使用了两个额外的指针 x 和 y，占用 O(1)的空间。

因此，总空间复杂度是 O(n)，主要由链表中存储的节点数量决定。

5 总结与心得

本实验在写代码时，由于在部分位置没有考虑特殊情况导致了结果出错例如运行超时、循环未退出等等。因此，需要考虑到特殊情况，以增加代码的鲁棒性。

在设计顺序表的实现是，对于如果进行取模运算，以及索引如何改变进行了不断地研究、尝试和改进，最终完善了顺序表的代码。但在本题中，用顺序表实现第二问颇具难度，在尝试后仍然无法实现，因此只设计了第一问的解决方案。