《数据结构与算法实验》第 4 次实验

学院： 专业： 年级：
姓名： 学号： 日期： 2022 年 3 月 19 日

第一部分 算法设计实现

一、 实验目的

1. 学习掌握线性表，尤其是循环链表、双链表、循环双链表的相关知识，熟悉其实现方式。
2. 通过线性表的应用实现实验的设计和验证。

二、 实验内容

1. **循环链表设计实现（带头节点）**：参考实验书 p174 单链表的实现做法，实现循环链表。
   - 加入求线性表的长度等操作。
   - 给定测试数据，验证抛出异常机制。
   - 将链表的结点结构用结点类实现，仿照实验书 p175 的做法。
2. **双链表的设计实现（带头节点）**： 参考实验书 p174 单链表的实现做法，实现双链表。
   - 加入求线性表的长度等操作。
   - 给定测试数据，验证抛出异常机制。
   - 将链表的结点结构用结点类实现，仿照实验书 p175 的做法。
3. **循环双链表的设计实现（带头节点）**：参考实验书 p174 单链表的做法，实现循环双链表。
   - 加入求线性表的长度等操作。
   - 给定测试数据，验证抛出异常机制。
   - 将链表的结点结构用结点类实现，仿照实验书 p175 的做法。
4. **集合的运算（设计实验：单链表的应用）**：用有序单链表实现集合的判等、交、并、差。
   - 对集合中的元素用有序单链表进行存储。
   - 实现交、并、差等基本运算时，不能另外申请储存空间。
   - 充分利用单链表的有序性，要求算法有较好的时间性能。

三、 设计与编码

1. 本实验用到的理论知识

·循环链表：在单链表中，如果将终端结点的指针域由空指针改为指向头结点，就使整个单链表形成一个环，这种头尾相接的单链表称为循环单链表。为了使空表和非空表的处理一致，通常也附设一个头结点。本次实现中，我们采用指向终端结点的尾指针来指示循环链表。

·双链表：在循环链表中，虽然从任一结点出发可以扫描到其他结点，但要找到其前驱结点，则需要遍历整个循环链表。如果希望快速确定表中任一结点的前驱结点，可以在单链表的每个结点中再设置一个指向其前驱结点的指针域，这样就形成了双链表，其结点结构如图 2 所示，除了新增的 prior，其余内容其实和单链表相同。特别的，对于构造、删除、插入函数，处理的时候都要注意前指针和后指针的改变。

·循环双链表：在双链表的基础上，将头结点和尾结点链接起来也能构成循环双链表，这样，无论是插入还是删除操作，对链表中开始结点、终端结点和中间任意结点的操作过程相同。

·单链表逆置算法：改变指针 next 的指向。

·单链表去重算法：从头开始往后找，把后面相同的值都删掉。

·集合的交算法：对于已知的链表 LA、LB。逐个考察 LA 中的元素，如果它不在 LB 中，则将它从 LA 中删去。对每个元素遍历操作后得到的 LA 为交集。

·集合的并算法：对于已知的链表 LA， LB。逐个考察 LB 中的元素，如果它不在 LA 中，则将它插入 LA。对每个元素遍历操作后得到的 LA 为并集。

2. 算法设计

（1） 循环链表设计实现

·定义结构体为结点，设置数值和指向下一个元素的指针。

·定义单链表类，成员数据有尾指针（指向表的尾结点）。

·设计成员函数：构造函数（有参、无参）、析构函数、长度函数、位置函数、插入函数、删除函数、输出函数。

（2） 双链表的设计实现：

·定义结构体为结点，设置数值和指向前结点的指针、指向后结点的指针。

• 定义单链表类，成员数据有头指针（指向表头的空结点）。

• 设计成员函数：构造函数（有参、无参）、析构函数、长度函数、位置函数、插入函数、删除函数、输出函数。

（3） 循环双链表的设计实现：

• 定义结构体为结点，设置数值和指向前结点的指针、指向后结点的指针。

• 定义单链表类，成员数据有头指针（指向表头的空结点）、尾指针（指向表的尾结点）。

• 设计成员函数：构造函数（有参、无参）、析构函数、长度函数、位置函数、插入函数、删除函数、输出函数。

（4） 循环双链表的设计实现：

• 定义顺序表类，成员数据有头指针。

• 设计成员函数：构造函数（有参、无参）、析构函数、长度函数、位置函数、插入函数、删除函数、输出函数。友元函数有判断是否相等函数、并集函数、交集函数、差集函数。

3. 编码

(1)循环链表的设计实现- LinkList.h

```
#ifndef LinkList_H
#define LinkList_H

template <class DataType>
struct Node
{
    DataType data;
    Node<DataType> *next;
};

template <class DataType>
class LinkList
{
    public:
```

```
        LinkList( );
        LinkList(DataType a[ ], int n);
        ~LinkList( );
        int Locate(DataType x);
        void Insert(int i, DataType x);
        void Delete(int i);
        void PrintList( );
        int Length( );
    private:
        Node<DataType> *rear;
};
#endif
```

(1)循环链表的设计实现- LinkList.cpp

```
#include <iostream>
#include "LinkList.h"
using namespace std;

template <class DataType>
LinkList<DataType> :: LinkList( )
{
    Node<DataType> *first = new Node<DataType>;
    first->next = first;
    rear = first;
}

template <class DataType>
LinkList<DataType> :: LinkList(DataType a[ ], int n)
{
    Node<DataType> *first,*r, *s;
    first = new Node<DataType>;
    r = first;
    for (int i = 0; i < n; i++)
    {
        s = new Node<DataType>;
        s -> data = a[i];
        r -> next = s;
        r = s;
    }
    rear = r;
    rear -> next = first;
}
```

```
template <class DataType>
LinkList<DataType> :: ~LinkList( )
{
    Node<DataType> *q,*first;
    first = rear -> next;
    while (first != rear)
    {
        q = first;
        first = first->next;
        delete q;
    }
    delete first;
}

template <class DataType>
int LinkList<DataType> ::Length( )
{
    Node<DataType> *p = (rear -> next) -> next;
    int l=0;
    while(p!=rear -> next)
    {
        l++;
        p=p->next;
    }
    return l;
}

template <class DataType>
void LinkList<DataType> :: Insert(int i, DataType x)
{
    if(i > Length() + 1) throw i;
    Node<DataType> *p, *s;
    int count = 0;
    p = rear -> next;
    while (count < i-1)
    {
        p = p -> next;
        count++;
    }
    s=new Node<DataType>;
    s -> data = x;
    s -> next = p->next;
    p -> next = s;
    if(i==Length()) rear=s;
```

```
}

template <class DataType>
void LinkList<DataType> :: Delete(int i)
{
    if (i > Length()) throw i;
    Node<DataType> *p, *d;
    DataType x;
    int count = 0;
    p = rear -> next;
    while (count < i - 1)
    {
        p = p -> next;
        count++;
    }
    if(count==Length()-1) rear=p;
    d = p->next;
    p->next = d->next;
    delete d;
    return ;
}

template <class DataType>
int LinkList<DataType> :: Locate(DataType x)
{
    Node<DataType> *p = rear -> next -> next;
    int count = 1;
    while (p != rear -> next)
    {
        if (p->data == x)
            return count;
        p = p->next;
        count++;
    }
    return 0;
}

template <class DataType>
void LinkList<DataType> :: PrintList( )
{
    Node<DataType> *p = rear -> next -> next;
    while (p != rear -> next)
    {
        cout << p->data<<"  ";
```

```
        p = p->next;
    }
    cout<<endl;
}
```

(1)循环链表的设计实现- LinkList_main.cpp

```
#include<iostream>
#include"LinkList.cpp"
using namespace std;

int main( )
{
    int m,r[100],f;
//插入
    cout<<"请输入插入的数组长度 m（不超过 100）： ";
    cin>>m;
    cout<<endl<<"请依次输入元素： ";
    for (int i = 0; i < m; i++)
        cin>>r[i];
    LinkList<int> L(r, m),L2(r, m);
    L.PrintList( );
    try
    {
        int p,da;
        cout<<"请输入插入数据的位置： ";
        cin>>p;
        cout<<endl<<"请输入插入的具体数据： ";
        cin>>da;
        L.Insert(p,da);
        cout<<endl<<"在位置"<<p<<"插入数据"<<da<<"结果为:"<<endl;
        L.PrintList( );
    }
    catch (int k)
    {
        cout<<"Insert error! i="<<k<<endl;
    }
//查找
    cout<<endl<<"请输入要查找的数据： ";
    cin>>f;
    cout<<endl<<"数据"<<f<<"的位置为： "<<endl;
    cout<<L.Locate(f)<<endl;
//删除
    cout<<endl<<"删除前数据为： "<<endl;
```

```
    L.PrintList( );
    try
    {
        int p;
        cout<<endl<<"请输入想要删除的数据的位置：";
        cin>>p;
        L.Delete(p);
        cout<<endl<<"删除位置"<<p<<"的数据后为："<<endl;
        L.PrintList( );
    }
    catch (int k)
    {
        cout<<"Delete error! i="<<k<<endl;
    }
    cout<<endl<<"数组长度为："<<endl<<L.Length()<<endl;
    return 0;
}
```

(2)双链表的设计实现- LinkList.h

```
#ifndef LinkList_H
#define LinkList_H

template <class DataType>
struct Node
{
    DataType data;
    Node<DataType> *next,*prior;
};

template <class DataType>
class LinkList
{
    public:
        LinkList( );
        LinkList(DataType a[ ], int n);
        ~LinkList( );
        int Locate(DataType x);
        void Insert(int i, DataType x);
        void Delete(int i);
        void PrintList( );
        int Length( );
    private:
        Node<DataType> *first;
```

```
};
#endif
```

(2)双链表的设计实现- LinkList.cpp

```
#include <iostream>
#include "LinkList.h"
using namespace std;
template <class DataType>
LinkList<DataType> :: LinkList( )
{
    Node<DataType> *first = new Node<DataType>;
    first -> next = NULL;
    first -> prior = NULL;
}

template <class DataType>
LinkList<DataType> :: LinkList(DataType a[ ], int n)
{
    Node<DataType> *r, *s;
    first = new Node<DataType>;
    r = first;
    for (int i = 0; i < n; i++)
    {
        s = new Node<DataType>;
        s -> data = a[i];
        r -> next = s;
        s -> prior = r;
        r = s;
    }
    r -> next = NULL;
    first -> prior = r;
}

template <class DataType>
LinkList<DataType> :: ~LinkList( )
{
    Node<DataType> *q;
    while (first != NULL)
    {
        q = first;
        first = first->next;
        delete q;
    }
```

```
}

template <class DataType>
int LinkList<DataType> ::Length( )
{
    Node<DataType> *p = first -> next;
    int l=0;
    while(p!=NULL)
    {
        l++;
        p=p->next;
    }
    return l;
}

template <class DataType>
void LinkList<DataType> :: Insert(int i, DataType x)
{
    if(i > Length() + 1) throw i;
    Node<DataType> *p = first, *s;
    int count = 0;
    while (count != i-1)
    {
        p = p -> next;
        count++;
    }
    if (i == Length()+1)
    {
        s=new Node<DataType>;
        s -> data = x;
        p -> next = s;
        s -> prior = p;
        s -> next = NULL;
    }
    else
    {
        s=new Node<DataType>;
        s -> data = x;
        s -> prior = p;
        s -> next = p -> next;
        p -> next -> prior = s;
        p -> next = s;
    }
}
```

```
template <class DataType>
void LinkList<DataType> :: Delete(int i)
{
    if (i > Length()) throw i;
    Node<DataType> *p = first,*s, *d;
    int count = 0;
    while (count != i - 1)
    {
        p = p -> next;
        count++;
    }
    d = p -> next;
    p -> next = d -> next;
    d -> next -> prior = p;
    delete d;
}

template <class DataType>
int LinkList<DataType> :: Locate(DataType x)
{
    Node<DataType> *p = first -> next;
    int count = 1;
    while (p != NULL)
    {
        if (p->data == x)
            return count;
        p = p->next;
        count++;
    }
    return 0;
}

template <class DataType>
void LinkList<DataType> :: PrintList( )
{
    Node<DataType> *p = first -> next;
    while (p != NULL)
    {
        cout <<p->data<<"  ";
        p = p -> next;
    }
    cout<<endl;
}
```

(2)双链表的设计实现- LinkList_main.cpp

主函数与单链表的设计实现相同。

(3)循环双链表的设计实现（不带头节点）- LinkList.h

```
#ifndef LinkList_H
#define LinkList_H

template <class DataType>
struct Node
{
    DataType data;
    Node<DataType> *next,*prior;
};

template <class DataType>
class LinkList
{
    public:
        LinkList( );
        LinkList(DataType a[ ], int n);
        ~LinkList( );
        int Locate(DataType x);
        void Insert(int i, DataType x);
        void Delete(int i);
        void PrintList( );
        int Length( );
    private:
        Node<DataType> *first;
};
#endif
```

(3)循环双链表的设计实现（不带头节点）- LinkList.cpp

```
#include <iostream>
#include "LinkList.h"
using namespace std;

template <class DataType>
LinkList<DataType> :: LinkList( )
{
    first = new Node<DataType>;
```

```
    first -> next = first;
    first -> prior = first;
}

template <class DataType>
LinkList<DataType> :: LinkList(DataType a[ ], int n)
{
    Node<DataType> *r, *s;
    first = new Node<DataType>;
    r = first;
    for (int i = 0; i < n; i++)
    {
        s = new Node<DataType>;
        s -> data = a[i];
        r -> next = s;
        s -> prior = r;
        r = s;
    }
    r -> next = first;
    first -> prior = r;
}

template <class DataType>
LinkList<DataType> :: ~LinkList( )
{
    Node<DataType> *q,*rear = first -> prior;
    while (first != rear)
    {
        q = first;
        first = first -> next;
        delete q;
    }
    delete first;
}

template <class DataType>
int LinkList<DataType> ::Length( )
{
    Node<DataType> *p = first -> next;
    int l=0;
    while(p!=NULL)
    {
        l++;
        p=p->next;
```

```
    }
    return l;
}

template <class DataType>
void LinkList<DataType> :: Insert(int i, DataType x)
{
    if(i > Length() + 1) throw i;
    Node<DataType> *p = first, *s;
    int count = 0;
    while (count != i-1)
    {
        p = p -> next;
        count++;
    }
    s=new Node<DataType>;
    s -> data = x;
    s -> prior = p;
    s -> next = p -> next;
    p -> next -> prior = s;
    p -> next = s;
}

template <class DataType>
void LinkList<DataType> :: Delete(int i)
{
    if (i > Length()) throw i;
    Node<DataType> *p = first,*s, *d;
    int count = 0;
    while (count != i - 1)
    {
        p = p -> next;
        count++;
    }
    d = p -> next;
    p -> next = d -> next;
    d -> next -> prior = p;
    delete d;
}

template <class DataType>
int LinkList<DataType> :: Locate(DataType x)
{
    Node<DataType> *p = first -> next;
```

```
    int count = 1;
    while (p != NULL)
    {
        if (p->data == x)
            return count;
        p = p->next;
        count++;
    }
    return 0;
}

template <class DataType>
void LinkList<DataType> :: PrintList( )
{
    Node<DataType> *p = first -> next;
    while (p != NULL)
    {
        cout <<p->data<<"  ";
        p = p -> next;
    }
    cout<<endl;
}
```

(3)循环双链表的设计实现（不带头节点）- LinkList_main.cpp

主函数与单链表的设计实现相同。

(4)集合的运算-LinkList.h

```
#ifndef LinkList_H
#define LinkList_H
template <class DataType>
struct Node
{
      DataType data;
      Node<DataType> *next;
};
template <class DataType>
class LinkList
{
    public:
        LinkList( );
```

```
        LinkList(int a[],int n);
        ~LinkList( );
        int Length( );
        void Delete(DataType x);
        void PrintList();
        int Locate(DataType x);
        void Insert(DataType x);
        template <class T>
        friend int IsEqual(LinkList<T> &A, LinkList<T> &B);
        template <class T>
        friend void Interest(LinkList<T> &A, LinkList<T> &B);
        template <typename T>
        friend void unionList(LinkList<T> &A, LinkList<T> &B);
        template <typename T>
        friend void Difference(LinkList<T> &A, LinkList<T> &B);
    private:
        Node<DataType> *first;
};
#endif
```

(4)集合的运算-LinkList.cpp

```
#include <iostream>
#include "LinkList.h"
using namespace std;
template <class DataType>
LinkList<DataType> :: LinkList( )
{
    first = NULL;
}

template <class DataType>
LinkList<DataType> :: LinkList(int a[],int n)
{
    if(n==0)
    {
        first=NULL;
    }
    else
    {
        Node<DataType> *r, *s;
        first = new Node<DataType>;
        first->data =a[0];
        r = first;
```

```
        first->next = NULL;
        for (int i = 1; i < n; i++)
        {
            s = new Node<DataType>;
            s->data = a[i];
            r->next = s;
            r = s;
        }
        r->next = NULL;
    }
}

template <class DataType>
LinkList<DataType> :: ~LinkList( )
{
    Node<DataType> *q;
    while (first != NULL)
    {
        q = first;
        first = first->next;
        delete q;
    }
}

template <class DataType>
void LinkList<DataType> :: Delete(DataType x)
{
    Node<DataType> *p, *q,*before;
    if(first->data==x)
    {
        q=first;
        first=first->next;
        delete q;
        return ;
    }
    before=first;
    p=first->next;
    while(p!=NULL)
    {
        if(p->data==x)
        {
            before->next=p->next;
            delete p;
            return ;
```

```
        }
        before=p;
        p=p->next;
    }
}

template <class DataType>
void LinkList<DataType> :: PrintList()
{
    Node<DataType> *q=first;
    if(q==NULL)
    {
        cout<<"empty set"<<endl;
        return ;
    }
    while (q != NULL)
    {
        cout<<q->data<<" ";
        q=q->next;
    }
    cout<<endl;
}

template <class DataType>
int LinkList<DataType> :: Locate(DataType x)
{
    Node<DataType> *q=first;
    while (q!= NULL)
    {
        if(q->data==x)
            return 1;
        q=q->next;
    }
    return 0;
}

template <class DataType>
void LinkList<DataType> :: Insert(DataType x)
{
    Node<DataType> *s= new Node<DataType>;
    s->data=x;
    s->next=first;
    first=s;
}
```

```
template <class DataType>
int LinkList<DataType> ::Length( )
{
    Node<DataType> *p = first->next;
    int l=0;
    while(p!=NULL)
    {
        l++;
        p=p->next;
    }
    return l;
}

template <class T>
int IsEqual(LinkList<T> &A, LinkList<T> &B)
{
    Node<T> *q=B.first;
    while (q != NULL)
    {
        if(A.Locate(q->data)==0)
            return 0;
        q=q->next;
    }
    q=A.first;
    while (q != NULL)
    {
        if(B.Locate(q->data)==0)
            return 0;
        q=q->next;
    }
    return 1;
}

template <class T>
void Interest(LinkList<T> &A, LinkList<T> &B)
{
    Node<T> *q=B.first,*tp;
    while (q != NULL)
    {
        if(A.Locate(q->data)==0)
        {
            tp=q->next;
            B.Delete(q->data);
```

```
            q=tp;
            continue;
        }
        q=q->next;
    }
}

template <typename T>
void unionList(LinkList<T> &A, LinkList<T> &B)
{
    Node<T> *q=B.first;
    while (q != NULL)
    {
        if(A.Locate(q->data)==0)
        {
            A.Insert(q->data);
        }
        q=q->next;
    }
}

template <typename T>
void Difference(LinkList<T> &A, LinkList<T> &B)
{
    Node<T> *q=A.first,*tp;
    while (q != NULL)
    {
        if(B.Locate(q->data)==1)
        {
            tp=q->next;
            A.Delete(q->data);
            q=tp;
            continue;
        }
        q=q->next;
    }
}
```

(4)集合的运算-LinkList_main.cpp

```
#include<iostream>
#include<cstdio>
#include"LinkList.cpp"
using namespace std;
```

```
int main()
{
    int ta[1000],tn;
    cout<<"请输入集合 A 的长度：";
    cin>>tn;
    cout<<"请按顺序键入集合 A 的元素：";
    for (int i = 0; i < tn; i++)
        cin>>ta[i];
    LinkList<int> A1(ta,tn),A2(ta,tn),A3(ta,tn);

    cout<<"请输入集合 B 的长度：";
    cin>>tn;
    cout<<"请按顺序键入集合 B 的元素：";
    for (int i = 0; i < tn; i++)
        cin>>ta[i];
    LinkList<int> B1(ta,tn),B2(ta,tn),B3(ta,tn);
    //判断相等
    if (IsEqual(A1,B1) == 1)
        cout<<"A,B 集合相等"<<endl;
    else cout<<"A,B 集合不相等"<<endl;
    //求交集
    Interest(A1,B1);
    cout<<"A,B 的交集是："<<endl;B1.PrintList();
    //求并集
    unionList(A2,B2);
    cout<<"A,B 的并集是："<<endl;A2.PrintList();
    //求差集
    cout<<"A,B 的差集是："<<endl;
    if(A3.Length()>B3.Length())
    {
        Difference(A3,B3);
        A3.PrintList();
    }
    else
    {
        Difference(B3,A3);
        B3.PrintList();
    }
    return 0;
}
```

四、 运行与测试

1.循环链表的设计实现

```
请输入插入的数组长度m（不超过100）：10

请依次输入元素：1 2 3 4 5 6 7 8 9 10
1  2  3  4  5  6  7  8  9  10
请输入插入数据的位置：3
请输入插入的具体数据：100

在位置3插入数据100结果为:
1  2  100  3  4  5  6  7  8  9  10

请输入要查找的数据：7

数据7的位置为:
8

删除前数据为:
1  2  100  3  4  5  6  7  8  9  10

请输入想要删除的数据的位置：4

删除位置4的数据后为:
1  2  100  4  5  6  7  8  9  10

数组长度为:
10
请按任意键继续. . .
```

2.双链表的设计实现

3.循环双链表的设计实现

由于实验 1.2.3 只在程序的设计上有区别，而在结果的实现上完全一致，所以 2.3 输出的结果和 1.的也完全相同。

4.用单链表实现集合的运算

测试用例如下：

(1) A={1,2 3 ,8}, B={3,5}

```
请输入集合A的长度：4
请按顺序键入集合A的元素：1 2 3 8
请输入集合B的长度：2
请按顺序键入集合B的元素：3 5
A,B集合不相等
A,B的交集是：
3
A,B的并集是：
5 1 2 3 8
A,B的差集是：
1 2 8
请按任意键继续. . .
```

(2) A={2,3,4,5,6} , B={2}

```
请输入集合A的长度：5
请按顺序键入集合A的元素：2 3 4 5 6
请输入集合B的长度：1
请按顺序键入集合B的元素：2
A,B集合不相等
A,B的交集是：
2
A,B的并集是：
2 3 4 5 6
A,B的差集是：
3 4 5 6
请按任意键继续. . .
```

(3) A={2}, B={2,3,4}

```
请输入集合A的长度：1
请按顺序键入集合A的元素：2
请输入集合B的长度：3
请按顺序键入集合B的元素：2 3 4
A,B集合不相等
A,B的交集是：
2
A,B的并集是：
4 3 2
A,B的差集是：
3 4
请按任意键继续. . .
```

(4) A={1,2,3}, B={4,5}

```
请输入集合A的长度：3
请按顺序键入集合A的元素：1 2 3
请输入集合B的长度：2
请按顺序键入集合B的元素：4 5
A,B集合不相等
A,B的交集是：
empty set
A,B的并集是：
5 4 1 2 3
A,B的差集是：
1 2 3
请按任意键继续. . .
```

(5) A={1,2,3}, B={}

```
请输入集合A的长度：3
请按顺序键入集合A的元素：1 2 3
请输入集合B的长度：0
请按顺序键入集合B的元素：A,B集合不相等
A,B的交集是：
empty set
A,B的并集是：
1 2 3
A,B的差集是：
```

五、 总结与心得

通过这次实验，我对单双链表及循环链表和模板有了更深刻的理解和更成熟的运用。尽管上周已经做过实现集合的交与并的实验，这周将顺序表变成单链表的过程中依旧遇到了很多麻烦。在设计实现集合的简单运算中需要考虑很多细节，包括但不限于两个集合之间的顺序等等，虽然在我们自己进行计算的时候不会考虑这些，但是由于程序是严格遵守设计好的步骤进行的，所以先后顺序在这里会有非常大的影响，这也是人脑之于程序更加优越的地方。在设计差集的程序时这一点体现得尤其明显。我的编码中稍微偷了一点懒，先后顺序得调整在主函数中才实现，如果能够在设计成员函数中就完成这一步，整体的代码将会更加简洁一些。

第二部分 设计实验：约瑟夫环问题

一、问题描述

设有编号为 1,2,…,n 的 n(n>0)个人围成一个圈，每个人持有一个密码 m，从第 1 个人开始报数，报到 m 时停止报数，报 m 的人出圈，再从他的下一个人起重新报数，报到 m 时停止报数，报 m 的出圈……直到所有人全部出圈为止。当任意给定 n 和 m 后，求 n 个人出圈的次序。

二、基本要求

- 用顺序表或无头节点的循环链表做；
- 建立数据模型，确定储存结构；
- 对任意 n 个人，密码为 m，实现约瑟夫环问题；
- 出圈的顺序可以依次输出，也可以用一个数组存储。

三、算法设计

1.线性表预处理

给定总人数 n 之后，设定一个长度为 n 的线性表，其数据元素分别有编号 1,2,3,…,n。

2.约瑟夫环函数（模拟报数）

如何判断人是否在圈内：对于顺序表，用一个临时数组 b。对于第 i 个人，在圈内则 b[i]=1，不在圈内则 b[i]=0。对于单链表，每次出圈进行删除操作即可。

如何进行循环报数：对于顺序表，循环的变量 i（1≤i≤n）,当 i=n 时，再将 i 改为 1 即可。对于单链表，直接将指向最后一个的指针改为头指针

3.抽象数据类型设计

设计一个 C++类。

成员变量：每个人的编号，表的长度。

成员函数：无参构造函数（默认长度 0），有参构造函数（传入参数长度，预处理线性表），析构函数，删除函数，约瑟夫环函数。

四、代码实现

1. 设计头文件-SeqList.h

```
#ifndef SeqList_H
#define SeqList_H
const int MaxSize = 1000;
template <class DataType>
class SeqList
{
    public:
        SeqList( );
        SeqList(int n);
        ~SeqList( );
        void Joseph(int m);

    private:
        DataType data[MaxSize];
        int length;
};
#endif
```

2. 设计成员函数-SeqList.cpp

```
#include <iostream>
#include "SeqList.h"
#include <cstdio>
using namespace std;

template <class DataType>
SeqList<DataType>::SeqList()
{
    length = 0;
}

template <class DataType>
SeqList<DataType>::SeqList(int n)
{
    for (int i=0;i<n;i++)
        data[i]=i;
    length=n;
}
```

```
template <class DataType>
SeqList<DataType> :: ~SeqList(){ }

template <class DataType>
void SeqList<DataType> :: Joseph(int m)
{
    bool b[MaxSize]={0};
    int n=length,count=0;
    int i=0;
    while(length!=0)
    {
        i++;
        if(i==n+1) i=1;
        if(b[i]==1) continue;
        count++;
        if(count==m)
        {
            b[i]=1;
            cout<<i<<" ";
            length--;
            count=0;
        }
    }
}
```

3. 设计主函数-Josephus.cpp

```
#include<iostream>
using namespace std;
#include "SeqList.cpp"

int main()
{
    int n,m;
    cout<<"Input n(the total number of people)"<<endl;
    cin>>n;
    cout<<"Input m(the password)"<<endl;
    cin>>m;
    SeqList<int> a(n);
    a.Joseph(m);
    return 0;
}
```

五、运行测试

测试结果如图。

```
Input n(the total number of people)
100
Input m(the password)
15
15 30 45 60 75 90 5 21 37 53 69 85 1 18 35 52 70 87 4 23 41 59 78 96 14 34 55 74 94
 13 36 57 79 99 22 44 66 89 11 38 62 84 9 33 63 88 16 43 71 98 27 56 86 19 49 81 12
 48 82 20 54 93 29 68 7 50 95 39 80 28 76 26 77 32 92 47 6 67 40 3 73 58 31 17 8 2
10 25 51 72 24 83 64 65 100 97 61 46 91 42 请按任意键继续. . .
```

六、总结与心得

相较于集合的交与并的实验，约瑟夫环实验的设计较为简单，代码也不算长，但是实现的功能也比较单一。做这个实验只需要充分理解约瑟夫环的实现原理，写出伪代码，然后再根据伪代码编写程序即可。这个程序的设计实用性较强，很大程度上解放了人工计算的过程，更加方便人们对这一数学问题进行研究。