中国矿业大学计算机学院实验报告

课程名称	数据结构实践		实验名称	第二章编程练习	
班级	信息安全 2019-1 班	姓名	李春阳	学号	10193657
仪器组号			实验日期	2020.12.5	
实验报告要求：1.实验目的 2.实验内容（题目描述，源代码，运行截图，调试情况） 3.实验体会					

一、实验目的

1 熟悉并掌握线性表的逻辑结构、物理结构。

2 熟悉并掌握顺序表的存储结构、基本操作和具体的函数定义。

3 熟悉 VC++程序的基本结构，掌握程序中的用户头文件、实现文件和主文件之间的相互关系及各自的作用。

4 熟悉 VC++操作环境的使用以及多文件的输入、编辑、调试和运行的全过程。

二、实验内容

1、第一题

1.1 题目描述

实现课本中的顺序表模板类，在模板类中实现如下操作：
构造函数（参数为顺序表的容量）和析构函数
顺序表的输入和输出
引用型操作：Locate,getData,Search,Size，Length，IsFull，IsEmpty
加工型操作：setData，Insert，Remove
在 main 方法中以一串整数为例测试以上所有的操作。

1.2 设计思路

```
# 顺序表模板类

## 私有成员

### 数据

### 最大容量

### 长度

## 构造函数和析构函数

### 构造函数

- 最大容量构造
```

    - 先判断是否输入合法
    - 若合法动态构造数组，判断是否内存可用
    - 若非法输入报错

- 顺序表引用构造

    - 动态构造数组判断是否内存可用
    - 循环赋值

析构函数

- 删除释放数组内存

输入和输出

输入

- 输入个数控制输入循环
- 循环输入数组元素

输出

- 循环输出数组

引用型操作

Locate

- 检查第 i 个值是否存在

    - 取第 i 个值的位置

getData

- 得到第 i 个值返回 x

    - 取第 i 个值返回给 x

Search

- 查询 x 是否在表中
- 循环数组，判断数组值是否与输入值是否相等

Size

- 返回最大容量

Length

- 返回长度

IsFull

- 是否长度与最大容量相等

IsEmpty

- 是否长度为 0

加工型操作

setData

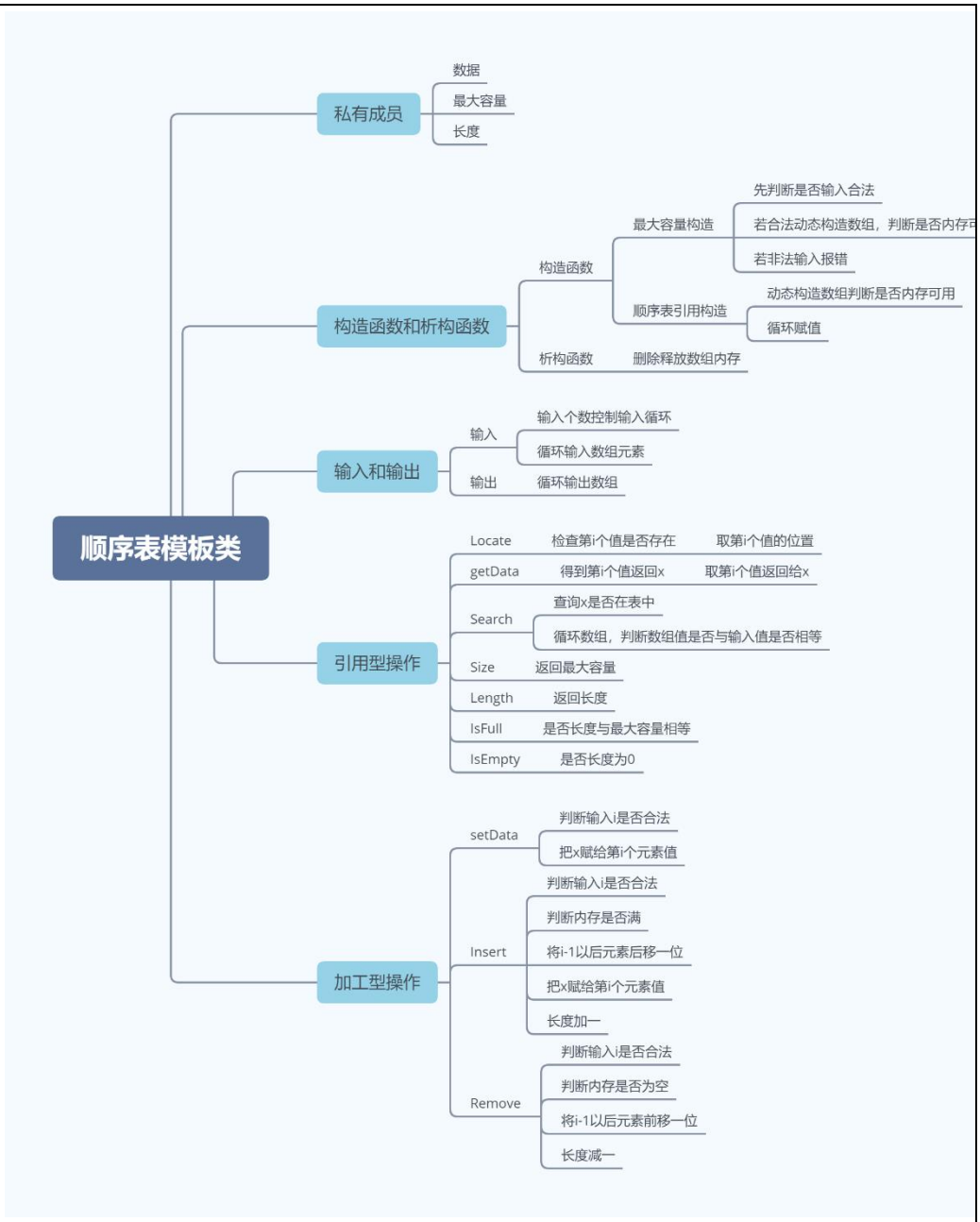
- 判断输入 i 是否合法
- 把 x 赋给第 i 个元素值

Insert

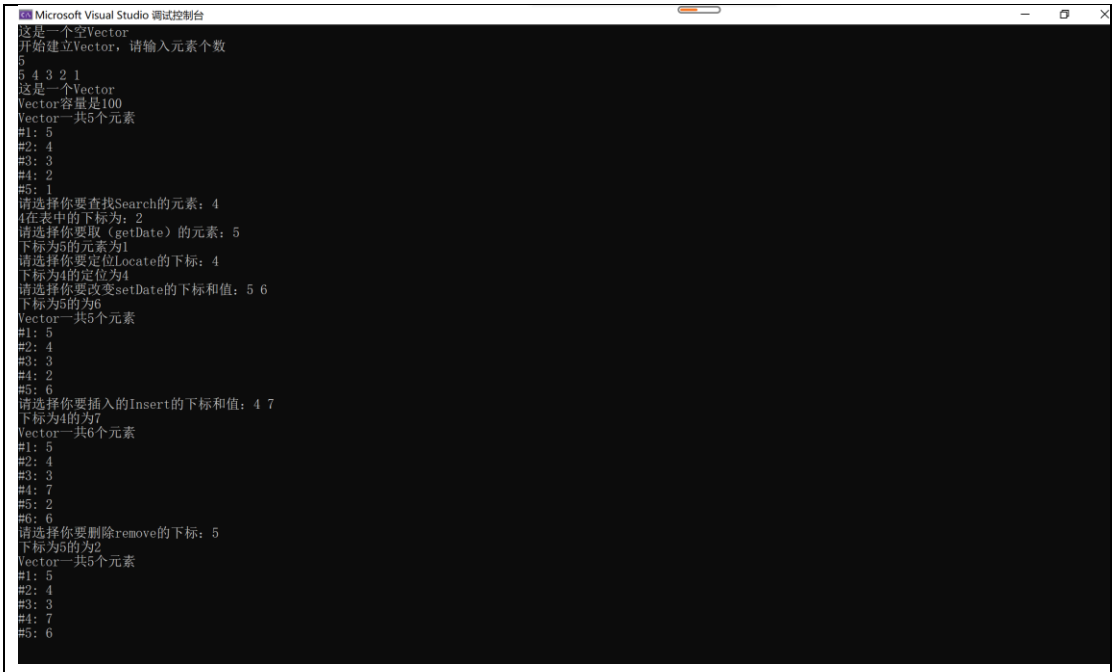
- 判断输入 i 是否合法
- 判断内存是否满
- 将 i-1 以后元素后移一位
- 把 x 赋给第 i 个元素值
- 长度加一

Remove

- 判断输入 i 是否合法
- 判断内存是否为空
- 将 i-1 以后元素前移一位
- 长度减一


顺序表模板类
私有成员
数据
最大容量
长度
构造函数和析构函数
构造函数
最大容量构造
先判断是否输入合法
若合法动态构造数组，判断是否内存可
若非法输入报错
顺序表引用构造
动态构造数组判断是否内存可用
循环赋值
析构函数
删除释放数组内存
输入和输出
输入
输入个数控制输入循环
循环输入数组元素
输出
循环输出数组
引用型操作
Locate
检查第i个值是否存在
取第i个值的位置
getData
得到第i个值返回x
取第i个值返回给x
Search
查询x是否在表中
循环数组，判断数组值是否与输入值是否相等
Size
返回最大容量
Length
返回长度
IsFull
是否长度与最大容量相等
IsEmpty
是否长度为0
加工型操作
setData
判断输入i是否合法
把x赋给第i个元素值
Insert
判断输入i是否合法
判断内存是否满
将i-1以后元素后移一位
把x赋给第i个元素值
长度加一
Remove
判断输入i是否合法
判断内存是否为空
将i-1以后元素前移一位
长度减一

1.3 运行截图



2、第二题

2.1 题目描述

实现课本中的带附加头结点的单链表模板类，完成如下功能：

定义链表节点的结构体类型

构造函数和析构函数

单链表的输入输出

引用型操作：getData,Locate ,Search,Length，IsEmpty, getHead

加工型操作：setData，Insert，Remove

在 main 方法中以一串整数为例测试以上所有的操作。

2.2 设计思路

顺序表模板类

私有成员

数据

最大容量

长度

构造函数和析构函数

构造函数

- 最大容量构造

    - 先判断是否输入合法
    - 若合法动态构造数组，判断是否内存可用
    - 若非法输入报错

- 顺序表引用构造

    - 动态构造数组判断是否内存可用
    - 循环赋值

析构函数

- 删除释放数组内存

输入和输出

输入

- 输入个数控制输入循环
- 循环输入数组元素

输出

- 循环输出数组

引用型操作

Locate

- 检查第 i 个值是否存在

    - 取第 i 个值的位置

getData

- 得到第 i 个值返回 x

    - 取第 i 个值返回给 x

Search

- 查询 x 是否在表中
- 循环数组，判断数组值是否与输入值是否相等

Size

- 返回最大容量

Length

- 返回长度

IsFull

- 是否长度与最大容量相等

IsEmpty

- 是否长度为 0

加工型操作

setData

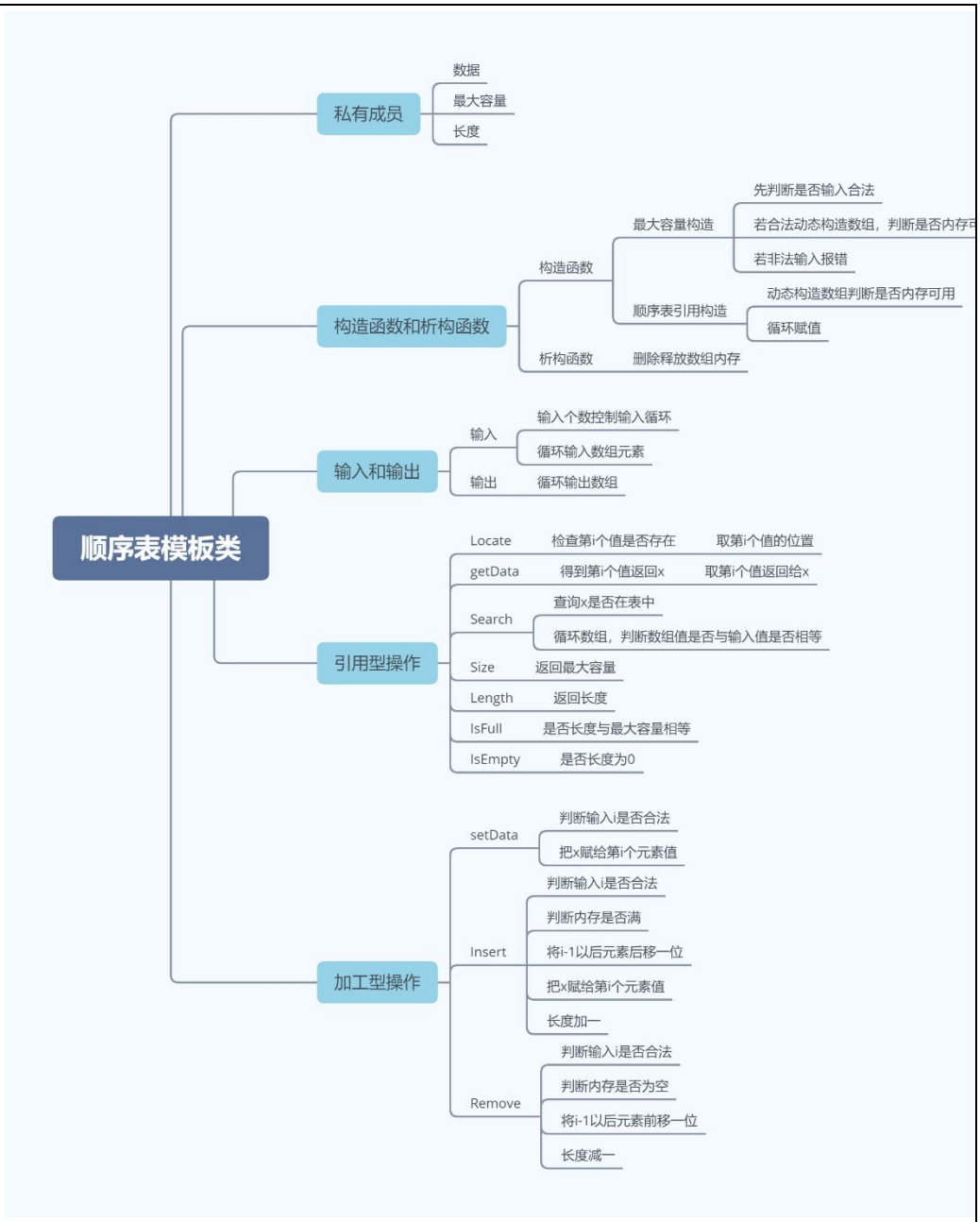
- 判断输入 i 是否合法
- 把 x 赋给第 i 个元素值

Insert

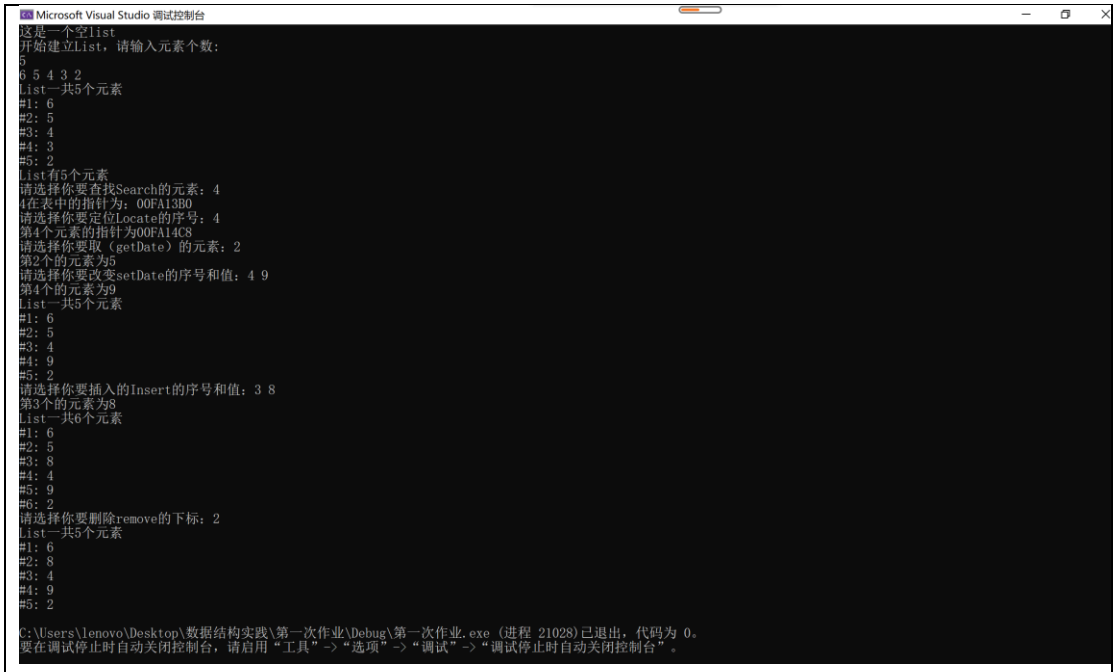
- 判断输入 i 是否合法
- 判断内存是否满
- 将 i-1 以后元素后移一位
- 把 x 赋给第 i 个元素值
- 长度加一

Remove

- 判断输入 i 是否合法
- 判断内存是否为空
- 将 i-1 以后元素前移一位
- 长度减一


顺序表模板类
私有成员
数据
最大容量
长度
构造函数和析构函数
构造函数
最大容量构造
先判断是否输入合法
若合法动态构造数组，判断是否内存可
若非法输入报错
顺序表引用构造
动态构造数组判断是否内存可用
循环赋值
析构函数
删除释放数组内存
输入和输出
输入
输入个数控制输入循环
循环输入数组元素
输出
循环输出数组
引用型操作
Locate
检查第i个值是否存在
取第i个值的位置
getData
得到第i个值返回x
取第i个值返回给x
Search
查询x是否在表中
循环数组，判断数组值是否与输入值是否相等
Size
返回最大容量
Length
返回长度
IsFull
是否长度与最大容量相等
IsEmpty
是否长度为0
加工型操作
setData
判断输入i是否合法
把x赋给第i个元素值
Insert
判断输入i是否合法
判断内存是否满
将i-1以后元素后移一位
把x赋给第i个元素值
长度加一
Remove
判断输入i是否合法
判断内存是否为空
将i-1以后元素前移一位
长度减一

2.3 运行截图



3、第三题

3.1 题目描述

利用基础题里构建的顺序表类创建两个有序的整数顺序表对象，实现将两个有序顺序表归并成一个新的有序顺序表并输出该新有序顺序表的结果。（可以调用已定义的顺序表类的方法来实现，并注意如何将两个有序的线性表进行归并的算法）

3.2 源代码

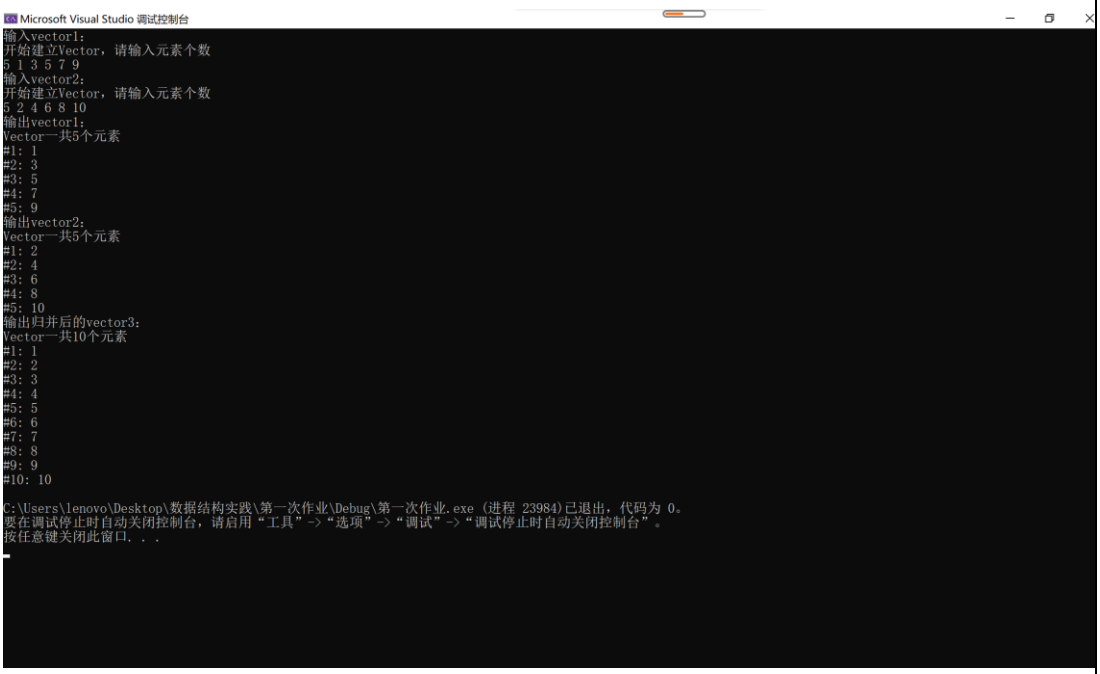
```
Vector<int> vector1(100);
    Vector<int> vector2(100);
    Vector<int> vector3(100);
    cout << "输入vector1：" << endl;
    vector1.input();
    cout << "输入vector2：" << endl;
    vector2.input();
    cout << "输出vector1：" << endl;
    vector1.output();
    cout << "输出vector2：" << endl;
    vector2.output();

    int i = 0, j = 1, k = 1;
    int x1 = 0, x2 = 0;
    for (i = 1; i <= vector1.Length() + vector2.Length(); i++)
    {
        if (j <= vector1.Length())
        {
            vector1.getDate(j, x1);
        }
```

```
        else
        {
            x1 = InfiniteDada;
        }
        if (k <= vector2.Length())
        {
            vector2.getDate(k, x2);
        }
        else
        {
            x2 = InfiniteDada;
        }
        if (x1 <= x2)
        {
            vector3.Insert(i, x1);
            j++;
        }
        else
        {
            vector3.Insert(i, x2);
            k++;
        }
    }

    cout << "输出归并后的vector3：" << endl;
    vector3.output();
```

3.3 运行截图



4、第四题

4.1 题目描述

利用基础题里构建的单链表类创建两个有序的整数链表对象，实现将两个有序链表归并成一个新的有序链表并输出该新有序链表的结果。（可以调用已定义的链表类的方法来实现，并注意如何将两个有序的线性表进行归并的算法）

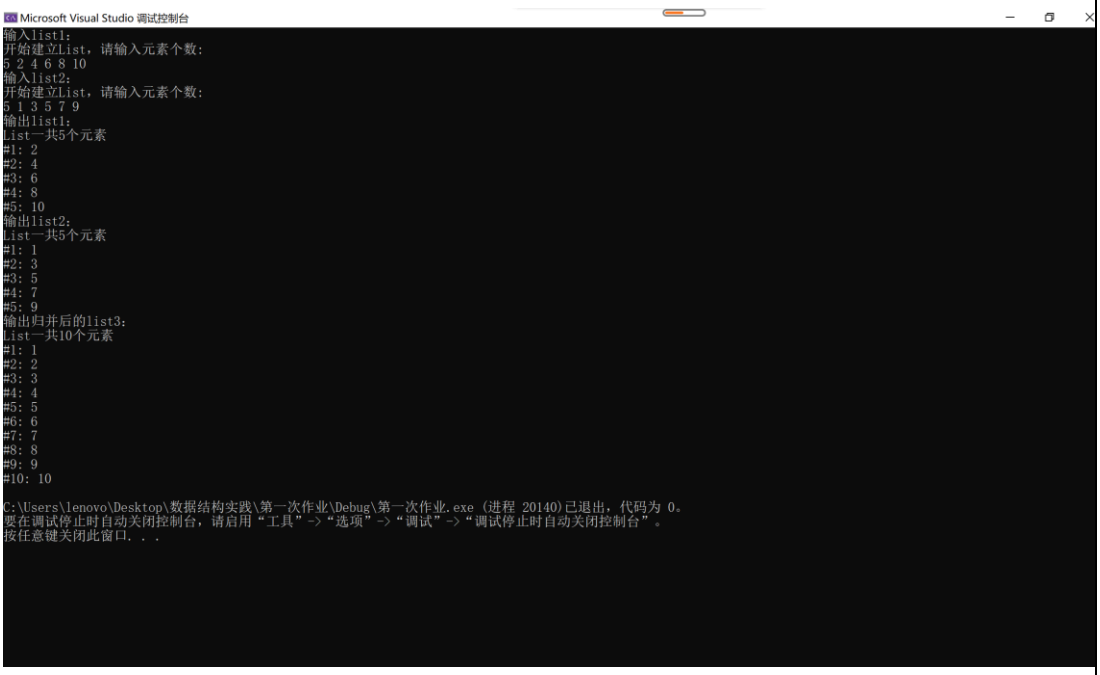
4.2 源代码

```
List<int> list1;
List<int> list2;
List<int> list3;
cout << "输入list1：" << endl;
list1.input();
cout << "输入list2：" << endl;
list2.input();
cout << "输出list1：" << endl;
list1.output();
cout << "输出list2：" << endl;
list2.output();

int i = 0, j = 1, k = 1;
int x1 = 0, x2 = 0;
for ( i = 1; i <= list1.Length() + list2.Length(); i++)
{
    if (j <= list1.Length())
    {
        list1.getDate(j, x1);
    }
    else
    {
        x1 = InfiniteDada;
    }
    if (k <= list2.Length())
    {
        list2.getDate(k, x2);
    }
    else
    {
        x2 = InfiniteDada;
    }
    if (x1 <= x2)
    {
        list3.Insert(i, x1);
        j++;
    }
    else
```

```
        {
            list3.Insert(i, x2);
            k++;
        }
    }
    cout << "输出归并后的list3：" << endl;
    list3.output();
```

4.3 运行截图



5、第五题

5.1 题目描述

编写一个求解 Josephus 问题的函数。用整数序列 1,2,3, ……,n 表示顺序围坐在圆桌周围的人。然后使用 n = 9, s = 1, m = 5，以及 n = 9, s = 1, m = 0，或者 n = 9, s = 1, m = 10 作为输入数据，检查你的程序的正确性和健壮性。最后分析所完成算法的时间复杂度。定义 JosephusCircle 类，其中含完成初始化、报数出圈成员函数、输出显示等方法。（可以选做其中之一，存储结构可以用循环链表或数组）

5.2 设计思路

构建一个循环列表，按 n-1 次循环，内部 m 次释放当前节点。

构建一个数组，在数组尾部建立为 flag，但为 flag 时下标返回 0，执行和上个列表后续操作一致

利用数学推论可知（数论入门）：f = (m + f) % i，则可快速求解

```
#define CPosi(T) CircLinkNode<T>*

template <typename T>    //结点定义
struct CircLinkNode
{
```

```
    T  data;           //结点数据
    CPosi(T) link;       //链接指针

CircLinkNode (CPosi(T) next = NULL ):link ( next ) { }
CircLinkNode ( T d,CPosi(T) next = NULL ):data(d), link(next) { }

};




template <typename T>
class JosephusCircle
{
private:
    CPosi(T) first;
    CPosi(T) last;

public:
    JosephusCircle();
    ~JosephusCircle();
    //获取头节点
    CPosi(T) getHead() {return first;};
    //
    CPosi(T) getTail() { return last;};
    bool insert (int, T& );
    CPosi(T) Locate(int);
    bool Remove(int, T&);

};
```

5.3 运行截图

三、程序附件

```
//数据结构第一次作业
#include <iostream>
#define InfiniteDada 2147483647

using namespace std;

/*****************************************************************
******************/
/**
 * 1. 基础题
 * （1）实现课本中的顺序表模板类，在模板类中实现如下操作：
 * 构造函数（参数为顺序表的容量）和析构函数
 * 顺序表的输入和输出
 * 引用型操作：Locate,getDate,Search, Size，Length，IsFull，IsEmpty
 * 加工型操作：setDate，Insert，Remove
 * 在 main 方法中以一串整数为例测试以上所有的操作。
 *
 */



/*****************************************************************
*****************************************************************/
//顺序表模板类

template<typename T>
class Vector
{
private:
```

```
    T* _date;                    //数据
    int _maxSize;                //最大容量
    int _size;                   //长度

public:
    Vector(int);
    Vector(Vector<T>&);
    ~Vector();

    //操作接口
    //容量查询
    int Size() const { return _maxSize;};
    //当前长度
    int Length() const { return _size;};
    //是否为空
    bool IsEmpty() {return _size == 0 ? true : false;};
    //是否已满
    bool IsFull() { return _size == _maxSize ? true : false;};
    //查询 x 是否在表中
    int Search(T&) const;
    //得到第 i 个值返回 x
    bool getDate(int,T&) const;
    //检查第 i 个值是否存在
    int Locate(int) const;
    //将第 i 个值修改为 x
    bool setDate(int,T&) const;
    //在第 i 个位置插入 x
    bool Insert(int, T&);
    //移除第 i 个位置的 x
    bool Remove(int, T&);
    //输入
    void input();
    //输出
    void output();

};

template<typename T> Vector<T>::Vector(int maxSize)
{
    if (maxSize > 0)
    {
        _maxSize = maxSize;
        _size = 0;
        _date = new T[_maxSize];
```

```
            if (_date == NULL)
            {
                cerr << "内存分配错误！" <<endl;
                exit(1);
            }
        }
        else
        {
            cerr << "内存分配错误！" <<endl;
            exit(1);
        }
}

template<typename T> Vector<T>::Vector(Vector<T>& V)
{
        _maxSize = V._maxSize;
        _size = V._size;
        T value;
        _date = new T[_maxSize];
        if (_date == NULL)
        {
            cerr << "内存分配错误！" <<endl;
            exit(1);
        }
        for (int i = 1; i < _size + 1; i++)
        {
            V.getDate(i,value);
            _date[i - 1] = value;
        }
}

template<typename T> Vector<T>::~Vector()
{
        delete[] _date;
}

template<typename T> int Vector<T>::Search(T& x) const
{
        for (int i = 0; i < _size + 1; i++)
        {
            if (_date[i] == x)
            {
                return i+1;
            }
```

```
	}
	return 0;
}

template<typename T> bool Vector<T>::getDate(int i,T& x) const
{
	if (i > 0 && i < _size + 1)
	{
		x = _date[i - 1];
		return true;
	}
	else
	{
		return false;
	}

}

template<typename T> int Vector<T>::Locate(int i) const
{
	if(i > 0 && i < _size + 1)
	{
		return i;
	}
	else
	{
		return 0;
	}
}

template<typename T> bool Vector<T>::setDate(int i, T& x) const
{
	if (i > 0 && i < _size + 1)
	{
		_date[i - 1] = x;
		return true;
	}
	else
	{
		return false;
	}
}

template<typename T> bool Vector<T>::Insert(int i, T& x)
```

```
{
	if (i < 0 && i > _size + 1)
	{
		return false;
	}
	if (_size == _maxSize)
	{
		//扩容
		return false;
	}
	for (int j = _size; j >= i ; j--)
	{
		_date[j] = _date[j - 1];
	}
	_date[i - 1] = x;
	_size++;
	return true;
}

template<typename T> bool Vector<T>::Remove(int i, T& x)
{
	if (i < 0 && i > _size + 1)
	{
		return false;
	}
	if (_size == 0)
	{
		return false;
	}
	x = _date[i - 1];
	for (int j = i; j < _size; j++)
	{
		_date[j - 1] = _date[j];
	}
	_size--;
	return true;
}

template<typename T> void Vector<T>::input()
{
	cout << "开始建立 Vector，请输入元素个数" << endl;
	while (true)
	{
		cin >> _size;
```

```
            if (_size <= _maxSize)
            {
                break;
            }
            cout << "输入有误，范围不超过" << _maxSize << endl;
        }
        for (int i = 0; i < _size ; i++)
        {
            cin >> _date[i];
        }
}

template<typename T> void Vector<T>::output()
{
        cout << "Vector 一共" << _size << "个元素" << endl;
        for (int i = 0; i < _size; i++)
        {
            cout << "#" << i + 1 << ": " << _date[i] << endl;
        }
}


/*****************************************************************
*****************************************************************/

/**
 * （2）实现课本中的带附加头结点的单链表模板类，完成如下功能：
 * 定义链表节点的结构体类型
 * 构造函数和析构函数
 * 单链表的输入输出
 * 引用型操作：getDate,Locate ,Search,Length，IsEmpty, getHead
 * 加工型操作：setDate，Insert，Remove
 * 在 main 方法中以一串整数为例测试以上所有的操作。
 */



/*****************************************************************
*****************************************************************/
//带附加头结点的单链表模板类

#define Posi(T) ListNode<T>*
```

```
template <class T>    struct ListNode
{
      T date;                                  //数值
      Posi(T) link;                           //后继指针


      //默认构造器
      ListNode(T item, Posi(T) ptr = NULL)
      {
            date = item;
            link = ptr;
      }
      ListNode(Posi(T) ptr = NULL)
      {
            link = ptr;
      };
};

template <typename T>
class List
{
private:
      Posi(T) first;            //头节点
      int _size;                  //长度


public:
      //构造函数
      List();
      List(const T&);
      List(List<T>&);
      //析构函数
      ~List();

      //操作接口
      //置空，把 List 清空
      void makeEmpty();
      //判断是否为空 List
      bool IsEmpty()
      {
            if (_size == 0)
            {
                  return true;
            }
```

```
            return false;
            //_size == 0 ? true : false;
        };
        //获得头节点
        Posi(T) getHead() const {return first;};
        //获取长度
        int Length() const {return _size;};
        //查找 x 的位置，返回 x 所在指针
        Posi(T) Search(T);
        //定位第 i 个元素的指针
        Posi(T) Locate(int);
        //获取第 i 个元素 x
        bool getDate(int, T&);
        //修改第 i 个元素内容
        bool setDate(int, T&);
        //在第 i 个元素处插入 x
        bool Insert(int, T&);
        //移除第 i 个元素
        bool Remove(int, T&);
        //输入
        void input();
        //输出
        void output();
};


template <typename T> List<T>::List()
{
        _size = 0;
        first = new ListNode<T>;
}

template <typename T> List<T>::List(const T& x)
{
        _size = 0;
        first = new ListNode<T>;
        (*this).Insert(0, x);
}

template <typename T> List<T>::List(List<T>& list)
{
        _size = list.Length();
        T value;
        Posi(T) srcptr = list.getHead();
```

```
	first = new ListNode<T>;
	Posi(T) p = first;
	while (srcptr->link != NULL)
	{
		value = srcptr->link->date;
		p->link = new ListNode<T>(value);
		p = p->link;
		srcptr = srcptr->link;
	}
	p->link = NULL;
}


template <typename T> List<T>::~List()
{
	makeEmpty();
}

template <typename T> void List<T>::makeEmpty()
{
	Posi(T) p;
	while (first->link != NULL)
	{
		p = first->link;
		first->link = p->link;
		delete p;
	}
	_size = 0;
}


template <typename T> Posi(T) List<T>::Search(T x)
{
	Posi(T) p = first->link;
	while (p != NULL)
	{
		if (p->date == x)
		{
			break;
		}
		else
		{
			p = p->link;
		}
```

```
	}
	return p;
}

template <typename T> Posi(T) List<T>::Locate(int i)
{
	if (i < 0 || i > _size + 1)
	{
		return NULL;
	}
	Posi(T) p = first;
	int j = 0;
	while (p != NULL && j < i)
	{
		p = p->link;
		j++;
	}
	return p;
}

template <typename T> bool List<T>::getDate(int i,T& x)
{
	if (i <= 0 || i > _size + 1)
	{
		return false;
	}
	Posi(T) p = Locate(i);
	if (p == NULL)
	{
		return false;
	}
	else
	{
		x = p->date;
		return true;
	}
}


template <typename T> bool List<T>::setDate(int i, T& x)
{
	if (i < 0 && i > _size + 1)
	{
		return false;
```

```
    }
    Posi(T) p = Locate(i);
    p->date = x;
    return true;
}


template <typename T> bool List<T>::Insert(int i, T& x)
{

    Posi(T) p = Locate(i - 1);
    if (p == NULL)
    {
        return false;
    }
    Posi(T) newNode = new ListNode<T>(x);
    if (newNode == NULL)
    {
        cerr << "内存分配错误！" << endl;
        exit(1);
    }
    newNode->link = p->link;
    p->link = newNode;
    _size++;
    return true;
}

template <typename T> bool List<T>::Remove(int i, T& x)
{
    Posi(T) p = Locate(i - 1);
    if (p == NULL || p->link == NULL)
    {
        return false;
    }
    Posi(T) del = p->link;
    p->link = del->link;
    x = del->date;
    delete del;
    _size--;
    return true;
}


template <typename T> void List<T>::output()
```

```
{
	int i = 0;
	Posi(T) p = first->link;
	cout << "List 一共" << _size << "个元素" << endl;
	while (p != NULL)
	{
		cout << "#" << i + 1 << ": " << p->date << endl;
		p = p->link;
		i++;
	}
}

template <typename T> void List<T>::input()
{
	Posi(T) newNode;
	Posi(T) p;
	T x;
	cout << "开始建立 List，请输入元素个数:" << endl;
	while (true)
	{
		cin >> _size;
		if (_size > 0)
		{
			break;
		}
		cout << "输入有误" << endl;
	}
	cin >> x;
	newNode = new ListNode<T>(x);
	first->link = newNode;
	for (int i = 1; i < _size; i++)
	{
		cin >> x;
		newNode = new ListNode<T>(x);
		if (newNode == NULL)
		{
			cerr << "内存分配错误！" << endl;
			exit(1);
		}
		p = Locate(i);
		p->link = newNode;
	}

}
```

```
/*****************************************************************
*****************************************************************/

/*
int main(int argc, char const *argv[])
{
*/
	//测试 Vector
	/*
	int x = 0;
	int i = 0;
	Vector<int> V(100);
	if (V.IsEmpty())
	{
		cout << "这是一个空 Vector" << endl;
	}
	if (V.IsFull())
	{
		cout << "这是一个满 Vector" << endl;
	}
	if (!V.IsEmpty() && !V.IsFull())
	{
		cout << "这是一个 Vector" << endl;
	}
	V.input();
	if (!V.IsEmpty() && !V.IsFull())
	{
		cout << "这是一个 Vector" << endl;
	}
	cout << "Vector 容量是" << V.Size() << endl;
	V.output();
	cout << "请选择你要查找 Search 的元素：";
	cin >> x;
	cout << x << "在表中的下标为：" << V.Search(x) << endl;
	cout << "请选择你要取（getDate）的元素：";
	cin >> i;
	V.getDate(i, x);
	cout << "下标为" << i << "的元素为" << x << endl;
	cout << "请选择你要定位 Locate 的下标：";
```

```
cin >> i;
x = V.Locate(i);
cout << "下标为" << i << "的定位为" << x << endl;
cout << "请选择你要改变 setDate 的下标和值：";
cin >> i >> x;
V.setDate(i, x);
cout << "下标为" << i << "的为" << x << endl;
V.output();
cout << "请选择你要插入的 Insert 的下标和值：";
cin >> i >> x;
V.Insert(i, x);
cout << "下标为" << i << "的为" << x << endl;
V.output();
cout << "请选择你要删除 remove 的下标：";
cin >> i;
V.Remove(i, x);
cout << "下标为" << i << "的为" << x << endl;
V.output();

*/
//测试 List
/*
int x = 0;
int i = 0;
List<int> list;
if (list.IsEmpty())
{
	cout << "这是一个空 list" << endl;
}
list.input();
list.output();
cout << "List 有" << list.Length() << "个元素" << endl;
cout << "请选择你要查找 Search 的元素：";
cin >> x;
cout << x << "在表中的指针为：" << list.Search(x) << endl;
cout << "请选择你要定位 Locate 的序号：";
cin >> i;
cout << "第" << i << "个元素的指针为" << list.Locate(i) << endl;
cout << "请选择你要取（getDate）的元素：";
cin >> i;
list.getDate(i,x);
cout << "第" << i << "个的元素为" << x << endl;
cout << "请选择你要改变 setDate 的序号和值：";
cin >> i >> x;
```

```
list.setDate(i, x);
cout << "第" << i << "个的元素为" << x << endl;
list.output();
cout << "请选择你要插入的 Insert 的序号和值：";
cin >> i >> x;
list.Insert(i, x);
cout << "第" << i << "个的元素为" << x << endl;
list.output();
cout << "请选择你要删除 remove 的下标：";
cin >> i;
list.Remove(i, x);
list.output();
*/



//提高题
/*
Vector<int> vector1(100);
Vector<int> vector2(100);
Vector<int> vector3(100);
cout << "输入 vector1：" << endl;
vector1.input();
cout << "输入 vector2：" << endl;
vector2.input();
cout << "输出 vector1：" << endl;
vector1.output();
cout << "输出 vector2：" << endl;
vector2.output();

int i = 0, j = 1, k = 1;
int x1 = 0, x2 = 0;
for (i = 1; i <= vector1.Length() + vector2.Length(); i++)
{
	if (j <= vector1.Length())
	{
		vector1.getDate(j, x1);
	}
	else
	{
		x1 = InfiniteDada;
	}
	if (k <= vector2.Length())
```

```
        {
            vector2.getDate(k, x2);
        }
        else
        {
            x2 = InfiniteDada;
        }
        if (x1 <= x2)
        {
            vector3.Insert(i, x1);
            j++;
        }
        else
        {
            vector3.Insert(i, x2);
            k++;
        }
    }

    cout << "输出归并后的 vector3： " << endl;
    vector3.output();
    */

    //list 实现归并
     /*
    List<int> list1;
    List<int> list2;
    List<int> list3;
    cout << "输入 list1： " << endl;
    list1.input();
    cout << "输入 list2： " << endl;
    list2.input();
    cout << "输出 list1： " << endl;
    list1.output();
    cout << "输出 list2： " << endl;
    list2.output();

    int i = 0, j = 1, k = 1;
    int x1 = 0, x2 = 0;
    for ( i = 1; i <= list1.Length() + list2.Length(); i++)
    {
        if (j <= list1.Length())
        {
            list1.getDate(j, x1);
```

```
            }
            else
            {
                x1 = InfiniteDada;
            }
            if (k <= list2.Length())
            {
                list2.getDate(k, x2);
            }
            else
            {
                x2 = InfiniteDada;
            }
            if (x1 <= x2)
            {
                list3.Insert(i, x1);
                j++;
            }
            else
            {
                list3.Insert(i, x2);
                k++;
            }
        }
        cout << "输出归并后的 list3：" << endl;
        list3.output();
    */
    /*
        return 0;
}*/
```

```
/**********************************************************************
**********************************************************************/
```

/**编写一个求解 Josephus 问题的函数。用整数序列 1, 2, 3, ……, n 表示顺序围坐在圆桌周围的人。

```
 * 然后使用 n = 9, s = 1, m = 5，以及 n = 9, s = 1, m = 0，
 * 或者 n = 9, s = 1, m = 10 作为输入数据，检查你的程序的正确性和健壮性。
 * 最后分析所完成算法的时间复杂度。定义 JosephusCircle 类，其中含完成初始化、报数出圈成员函数、输出显示等方法。
 * （可以选做其中之一，存储结构可以用循环链表或数组）
 */



/*************************************************************************
*************************************************************************/

#define CPosi(T) CircLinkNode<T>*

template <typename T>       //结点定义
struct CircLinkNode
{
      T    data;                  //结点数据
      CPosi(T) link;            //链接指针

CircLinkNode (CPosi(T) next = NULL ):link ( next ) { }
CircLinkNode ( T d,CPosi(T) next = NULL ):data(d), link(next) { }

};




template <typename T>
class JosephusCircle
{
private:
      CPosi(T) first;
      CPosi(T) last;

public:
      JosephusCircle();
      ~JosephusCircle();
      //获取头节点
      CPosi(T) getHead() {return first;};
      //
      CPosi(T) getTail() { return last;};
      bool insert (int, T& );
```

```
    CPosi(T) Locate(int);
    bool Remove(int, T&);

};

template <typename T> JosephusCircle<T>::JosephusCircle()
{
    first = new CircLinkNode<T>();
    last = new CircLinkNode<T>();
    first->link = last;
    last->link = first;
}

template <typename T> JosephusCircle<T>::~JosephusCircle()
{
}

template <typename T> bool JosephusCircle<T>::insert(int i, T& x)
{
    if (i < 0)
    {
        return false;
    }
    CPosi(T) p = Locate(i);
    if (p == NULL)
    {
        return false;
    }
    CPosi(T) newNode = new CircLinkNode<T>(x);
    if (newNode == NULL)
    {
        cerr << "内存分配错误！"<< endl;
        exit(1);
    }
    newNode->link = p->link;
    p->link = newNode;
    return true;
}

template <typename T>    CPosi(T) JosephusCircle<T>::Locate(int i)
{
    if (i < 0 )
    {
        return NULL;
```

```
	}
	CPosi(T) p = first;
	int j = 0;
	while (p != NULL && j < i && p->link != last)
	{
		p = p->link;
		j++;
	}
	return p;
}


template <typename T> bool JosephusCircle<T>::Remove(int i, T& x)
{
	CPosi(T) p = Locate(i - 1);
	if (p == NULL || p->link == NULL)
	{
		return false;
	}
	CPosi(T) del = p->link;
	p->link = del->link;
	x = del->date;
	delete del;
	return true;
}


template <typename T>
void Josephus(JosephusCircle<T>& josephus, int n, int m)
{
	CPosi(T) p = josephus.getHead()->link;
	CPosi(T) pre = NULL;
	if (p == josephus.getTail())
	{
		//表为空
		exit(1);
	}
	int i, j, x = 0;
	for (i = 0; i < n - 1; i++)
	{
		for (j = 1; j < m ; j++)
		{
			pre = p;
			p = p->link;
```

```
                if (p == josephus.getTail())
                {
                    pre = josephus.getHead();
                    p = pre->link;
                }
            }
            cout << "第" << i + 1 << "轮出列的人为： " << p->data << endl;
            pre->link = p->link;
            delete p;
            p = pre->link;
        }
        cout << "最后留下的人是: " << josephus.getHead()->link->data << endl;

}



int main(int argc, char const *argv[])
{
        JosephusCircle<int> list;
        int i,n,m;
        cout << "输入游戏者人数和报数间隔 : ";
        cin >> n >> m;
        for ( i = 1; i <= n; i++)
        {
            list.insert(i - 1,i);
        }
        Josephus(list,n,m);

        return 0;
}



/*
//Josephus 问题的算法最终版：利用数学数论，递归循环求递推式达到算法优化。
#include <iostream>
#include <cmath>

using namespace std;

int lastRemaining(int n, int m)
{
```

```
	int f = 0;
	for (int i = 2; i != n + 1; ++i)
	{
		f = (m + f) % i;
	}
	return f;
}

int main()
{
	int n, result, m;
	cout << "输入游戏者人数和报数间隔 :";
	cin >> n >> m;
	result = lastRemaining(n, m);
	cout << result + 1 << endl;
	return 0;
}
*/
```