# 实验五

## 1. 问题定义及需求分析

### 1.1 问题描述

##### Josephus排列问题定义如下：假设n个竞赛者排成一个环形。给定一个正整数m≤n，从第1人开始，沿环计数，第m人出列。这个过程一直进行到所有人都出列为止。最后出列者为优胜者。全部出列次序定义了1，2，…n的一个排列。称为（n，m）Josephus排列。例如，（7，3）Josephus排列为3,6,2,7,5,1,4。

### 1.2 实验要求

##### 设计求解Josephus排列问题程序。

###### (1)采用顺序表、单链表或双向循环链表等数据结构。

###### (2)采用双向循环链表实现Josephus排列问题，且奇数次顺时针轮转，偶数次逆时针轮转。

###### (3)推荐采用静态链表实现Josephus排列问题。

###### (4)实现STL的双向循环链表类。

###### (5)双向循环链表类的简单应用。

### 1.3 输入

#### 1.3.1 输入的形式

##### 两个整数，n和m，定义同题中

#### 1.3.2 输入值的范围

##### 

### 1.4 输出形式

##### 一个由n个数字组成的序列，按照出列先后排序。

### 1.5 程序的功能

##### 输出Josephus排列，且若m为奇数则顺时针旋转，若m为偶数则逆时针旋转。

### 1.6 测试数据

#### 1.6.1 输入

##### (1) 10 3

##### (2) 3 6 9 2 7 1 8 5 10 4

#### 1.6.2 输出

##### (1) 13 4

##### (2) 11 7 3 12 6 1 8 13 4 5 2 9 10

## 2.概要设计

### 2.1 抽象数据类型定义（抄书上的ADT，只抄用到的函数就行）

### 2.2 主程序流程

##### (0)初始化变量

##### (1)在屏幕上展示问题情境

##### (2)处理输入数据

##### (3)把所有人都加入循环链表

##### (4)判断m的奇偶

##### (5)根据上一步的判断来执行循环，模拟报数并出圈

### 2.3 模块调用关系

## 3.详细设计

### 3.1 定义数据类型及存储结构

class CList//自定义的循环链表类  
{  
private:  
 std::list<int>Data;//底层是线性链表  
 std::list<int>::iterator it;//指针  
public:  
 void push\_back(int x);//加入新元素  
 int size();//返回元素个数  
 void erase\_prev();//删除指针所指向的元素，并让指针指向上一个元素  
 void erase\_next();//删除指针所指向的元素，并让指针指向下一个元素  
 void prev();//让指针指向上一个元素  
 void next();//让指针指向下一个元素  
 int get();//获取指针当前指向的元素  
};

### 3.2 每个函数及操作的代码

void CList::push\_back(int x)//加入新元素  
{  
 Data.push\_back(x);//把元素加入底层的线性链表  
 it=Data.begin();//让指针指向队首  
}  
int CList::size()//返回元素个数  
{  
 return Data.size();//返回线性链表的元素个数  
}  
void CList::erase()//删除指针所指向的元素，并后移指针  
{  
 it=Data.erase(it);//删除当前指针所指向的元素，并后移指针  
 if(it==Data.end())//如果到了队尾  
 it=Data.begin();//如果  
}  
void CList::next()//让指针指向下一个元素  
{  
 //说明：STL的list中的.end()迭代器中无元素，  
 //所以如果自增以后来到Data.end()，为了符合循环链表的性质，  
 //我们应让它指向队首元素。  
 ++it;//尝试自增迭代器  
 if(it==Data.end())//如果已经来到队尾  
 it=Data.begin();//那就指向队首  
 //如果没到队尾，什么都不做。  
}  
int CList::get()//获取指针当前指向的元素  
{  
 return \*it;//返回指针指向的元素  
}

## 4.调试分析

### 4.1 遇到的问题及分析

#### 4.1.1 问题

##### (1)在每一次报数后，如何知道上一轮谁出圈了？如何表达出圈人的位置？

##### (2)在删除元素的时候，总是发生数组越界导致程序崩溃。

#### 4.1.2 分析

##### (1)在自定义的CList类中加一个指针（STL迭代器），来记录当前操作的元素和它的位置。

##### (2)删除当前元素后，它的迭代器里面没有元素，因此在删除的时候还要把迭代器向前或向后移动。这里把删除分成了两个函数，erase\_prev()和erase\_next()，分别对应删除后把迭代器往前移动和往后移动。

### 4.2 算法时空分析

#### 4.2.1 时间

##### (1)添加元素

###### 由链表的性质，是O(1)

##### (2)删除元素

###### 由链表的性质，是O(1)

##### (3)总体算法时间复杂度

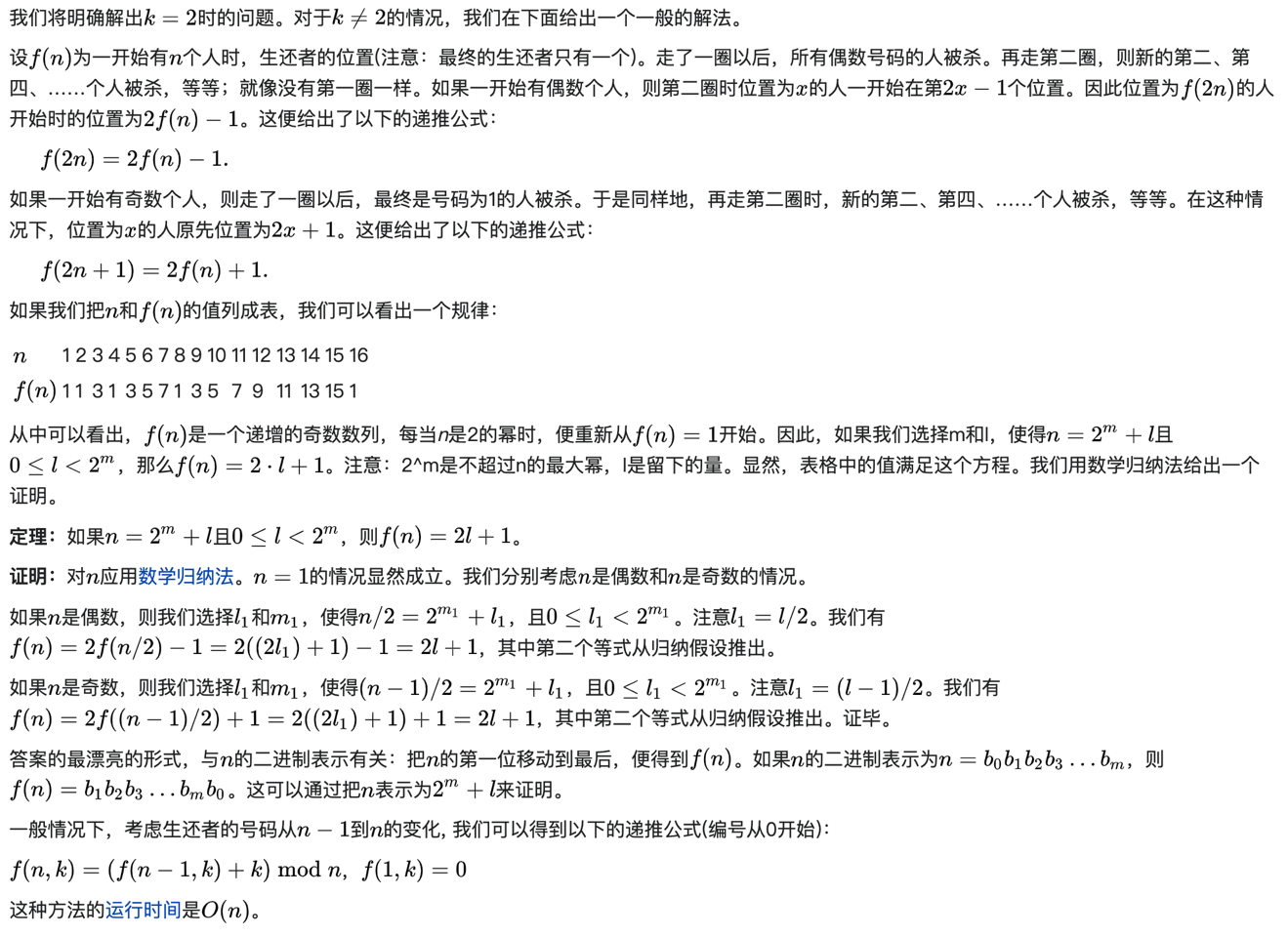
###### 因为一共有n人要出圈，而每个人出圈前要先经过m个人，因此总体的时间复杂度为O(n\*m)。

#### 4.2.2 空间

##### 要存n个人，所以是O(n)。

#### 4.2.3 改进

##### 在网上查阅资料得知，如果只想知道最后一个出圈的人，则可通过O(n)的时间复杂度解决。



### 4.3 经验和体会

##### (1)通过对双向循环链表的建立、遍历、删除等操作的实现，对链表和STL了解得更加透彻，掌握得更加牢固。

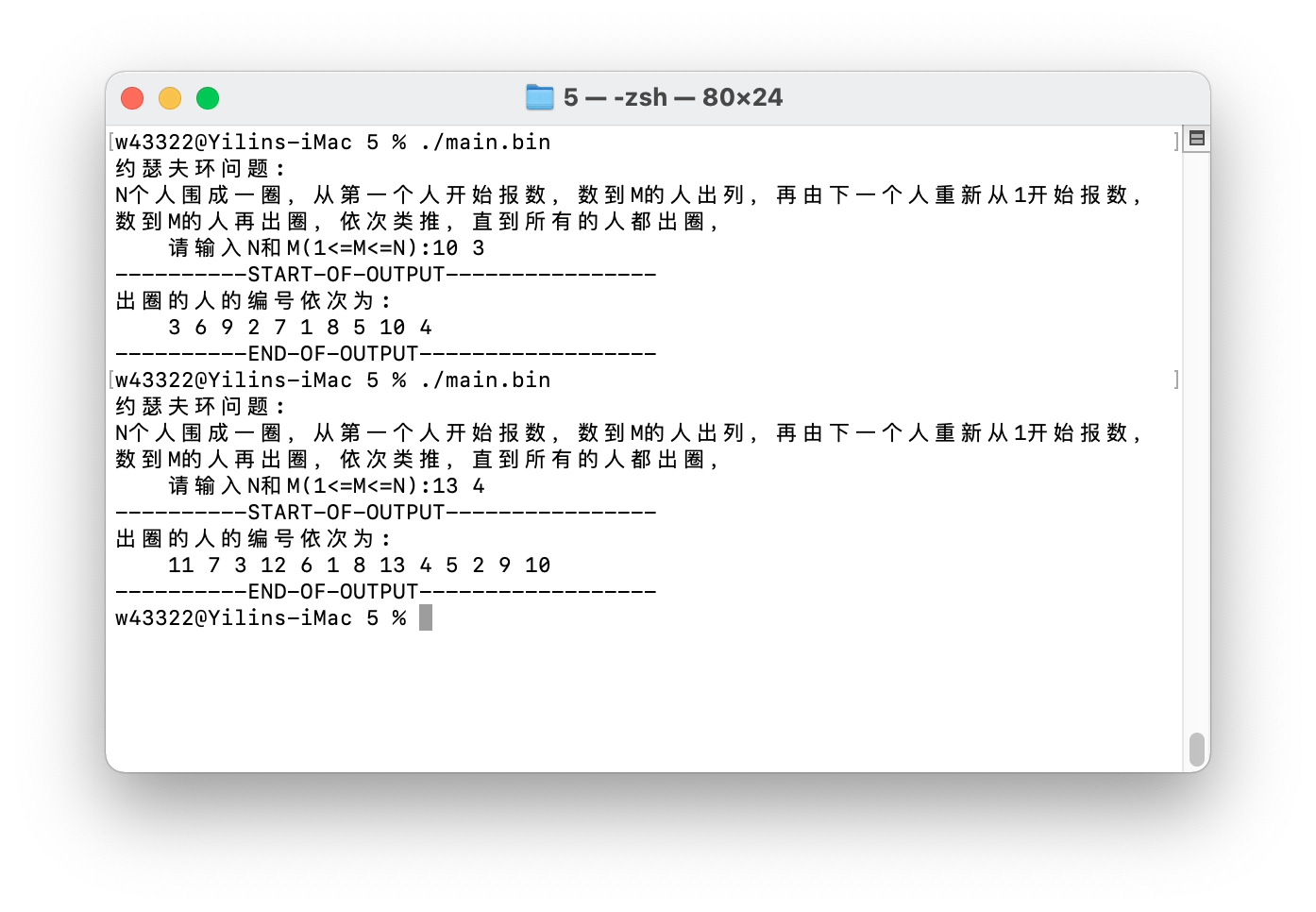
##### (2)对头结点问题的特殊处理，使自己解决问题的能力有了提升。

##### (3)对STL的类进行封装，锻炼了面向对象编程的能力。

## 5.使用说明

##### 按照提示输入n，m即可。

## 6.测试结果



## 7.附录

### 7.1 个人负责的部分

##### STL封装。

### 7.2 整个程序

#### 7.2.0 makefile

main : main.cpp clist.cpp  
 g++ -Wall -O2 -o main.bin main.cpp clist.cpp

#### 7.2.1.1 main.h

#ifndef MAIN\_H  
#define MAIN\_H  
  
#include<iostream>  
#include"clist.h"//自定义的循环链表类  
  
#endif //MAIN\_H

#### 7.2.1.2 main.cpp

#include"main.h"  
int main()  
{  
 int N,M;//初始化M和N  
 CList L;//初始化一个循环队列  
  
 //在屏幕上显示问题情境  
 std::cout<<"约瑟夫环问题:\n"  
 <<"N个人围成一圈，从第一个人开始报数，数到M的人出列，再由下一个人重新从1开始报数，\n"  
 <<"数到M的人再出圈，依次类推，直到所有的人都出圈，\n";  
 std::cout<<" 请输入N和M(1<=M<=N):";  
 std::cin>>N>>M;  
 while(1>M||M>N)//输入的数据不满足数据范围限制，提示重新输入  
 {  
 std::cout<<"输入错误，请重新输入!\n";  
 std::cout<<" 请输入N和M(1<=M<=N):";  
 std::cin>>N>>M;  
 }  
 std::cout<<"----------START-OF-OUTPUT----------------\n"  
 <<"出圈的人的编号依次为:\n ";  
  
 for(int i=1;i<=N;++i)  
 L.push\_back(i);//首先把所有人都加入循环链表  
   
 while(L.size())  
 {  
 //进入循环的条件是cnt>=1  
 for(int i=1;i<M;++i)  
 L.next();//此人不出圈，指针指向下一人  
 std::cout<<L.get()<<" ";//指针指向的人报的数是M，这个人应该出圈，输出这个人的编号  
 L.erase();//把这个人从链表中删除  
 }  
 std::cout<<"\n----------END-OF-OUTPUT------------------\n";  
}

#### 7.2.2.1 clist.h

#ifndef CLIST\_H  
#define CLIST\_H  
  
#include<list>//包含STL中的list  
class CList//自定义的循环链表类  
{  
private:  
 std::list<int>Data;//底层是线性链表  
 std::list<int>::iterator it;//指针  
public:  
 void push\_back(int x);//加入新元素  
 int size();//返回元素个数  
 void erase();//删除指针所指向的元素  
 void next();//让指针指向下一个元素  
 int get();//获取指针当前指向的元素  
};  
  
#endif //CLIST\_H

#### 7.2.2.2 clist.cpp

#include"clist.h"  
//本文件中包含CList中方法的实现  
void CList::push\_back(int x)//加入新元素  
{  
 Data.push\_back(x);//把元素加入底层的线性链表  
 it=Data.begin();//让指针指向队首  
}  
int CList::size()//返回元素个数  
{  
 return Data.size();//返回线性链表的元素个数  
}  
void CList::erase()//删除指针所指向的元素，并后移指针  
{  
 it=Data.erase(it);//删除当前指针所指向的元素，并后移指针  
 if(it==Data.end())//如果到了队尾  
 it=Data.begin();//如果  
}  
void CList::next()//让指针指向下一个元素  
{  
 //说明：STL的list中的.end()迭代器中无元素，  
 //所以如果自增以后来到Data.end()，为了符合循环链表的性质，  
 //我们应让它指向队首元素。  
 ++it;//尝试自增迭代器  
 if(it==Data.end())//如果已经来到队尾  
 it=Data.begin();//那就指向队首  
 //如果没到队尾，什么都不做。  
}  
int CList::get()//获取指针当前指向的元素  
{  
 return \*it;//返回指针指向的元素  
}