**项目说明文档**

**数据结构课程设计**

**——约瑟夫生者死者游戏**

作 者 姓 名 刘淑仪

学 号 2251730

指 导 教 师 张 颖

学 院 专 业 软件学院 软件工程



二〇二三 年 十 月 二十五 日

目录

1 项目分析 2

1.1 项目背景分析 2

1.2 项目功能分析 2

2 项目设计 2

2.1 数据结构设计 2

2.2 类结构设计 2

2.3 成员与操作设计 2

2.4 系统设计 4

3 功能设计与项目实现 4

3.1 总体设计功能的实现 4

3.1.1 总体系统实现思路 4

3.1.2 总体系统核心代码 5

3.1.3 总体功能实现示例 5

3.2 输入各类数字功能的实现 6

3.2.1 输入功能实现思路 6

3.2.2 输入功能核心代码 6

3.2.3 输入功能实现示例 7

3.3 约瑟夫环的核心算法的实现 8

3.3.1 约瑟夫环的核心算法实现思路 8

3.3.2 约瑟夫环的核心算法核心代码 8

3.3.3 约瑟夫环的核心算法实现示例 9

3.4 输出死者生者名单功能的实现 9

3.4.1 输出死者生者名单功能实现思路 9

3.4.2 输出死者生者名单功能核心代码 10

3.4.3 输出死者生者名单功能实现示例 12

3.5 本项目涉及CircList.h内容的主要实现 12

3.5.1 节点结构体和循环链表类的定义 12

3.5.2 CircList.h核心代码 12

4 项目测试 13

4.1 功能测试 13

4.1.1 基本功能测试 13

4.1.2 边界条件测试 16

4.2 错误测试 17

4.2.1 输入验证错误 17

4.2.1.1 输入验证错误判断思路 17

4.2.1.3输入示例 17

4.2.2 异常情况处理 19

4.3 Linux环境测试 19

5 集成开发环境与编译运行环境 20

# 项目分析

## 项目背景分析

约瑟夫生者死者游戏（Josephus Problem）是一个经典的数学和计算机科学问题，它涉及一个游戏，参与者按照一定规则逐个淘汰，直到最后剩下一个人。其具体内容如下：有n个人（编号从1到n）站成一个环，从编号1开始计数，每次数到第m个人，然后将他从环中淘汰出局，然后从下一个人开始重新数数。这个过程一直重复，直到只剩下一个人。问题的目标通常是找出最后幸存的那个人的编号。这个问题得名于古罗马犹太历史学家弗拉维奥·约瑟夫斯（Flavius Josephus），他据说在犹太战争中应用了这个策略来避免被罗马士兵俘虏。

Josephus问题是一个经典的抽象数学问题，它的应用不仅仅局限于游戏，还可以在计算机科学、算法设计以及数据结构的教育中找到实际用途。它有助于理解递归、循环链表、模运算等概念，并且在实际编程中也具有一定的挑战性。

解决这个问题的经典算法思路是使用循环链表和递归。这个问题的解决方式很直观，但在实现时需要小心处理链表节点的删除和指针的调整，以确保算法的正确性。Josephus问题是一个有趣的数学问题，也是学习数据结构和算法的好例子。

## 项目功能分析

约瑟夫生者死者游戏项目的功能是模拟一个环形排列的参与者，在每轮中淘汰指定数量的参与者，直到最后只剩下一个幸存者，然后找出该幸存者的编号。其项目的主要功能如下：创建环形排列的参与者、顺序淘汰参与者、重复游戏直到只剩下一个参与者、找出最后幸存者的编号。

其主要功能的实现主要依赖于

1、创建环形链表：用单循环链表来表示环形排列的参与者。链表的每个节点代表一个参与者，而且链表的"下一个"指针形成环。

2、顺序淘汰参与者：通过遍历链表，不断移动到下一个节点，以模拟游戏规则。在每一轮中，数m个节点，然后删除第m个节点。这可以通过适当的指针操作来实现。

3、重复游戏：游戏应该一直进行，直到链表中只剩下一个节点。这可以通过循环来实现，直到链表中的节点数为1。

4、找出最后幸存者的编号：一旦游戏结束，剩下的唯一节点即为最后幸存者。通过访问该节点的编号属性以获取最终答案。

# 项目设计

## 数据结构设计

由上文项目功能分析来看，本项目需要一个环形链表来存储排列的参与者且需要一个链表结点来链接整个链表结构。由于约瑟夫问题的本质是在一个环形排列的参与者中依次淘汰一定数量的人，直到只剩下一个人，而单循环链表天然地模拟了这种环形结构，因为链表的最后一个节点指向第一个节点，形成一个环。不仅如此，在单循环链表中，节点的删除操作非常高效，而且使用单循环链表简化了约瑟夫问题的算法实现。

## 类结构设计

首先在CircList.h中，创建CircList类来存储循环链表和其基本功能的实现。私有成员(private)为链表头结点和尾结点，公有成员(public)为实现本链表基本功能的各函数如构造函数、析构函数、计算链表长度、搜索对应值的元素的地址、删除某个元素、寻找链表中的最小值等。

然后在本项目的主要cpp(Josephus survivor game.cpp)中，创建Josephussolution类来存储JosephusList和各项参数的输入作为私有成员，公有成员则为实现本系统基本功能的各函数。

## 成员与操作设计

未注释的函数请见下文实现对应功能时代码的函数标签。

CircList.h内存储单循环链表、链表结点及其基本函数实现。

·存储循环链表的结点

template <class T>

struct CircLinkNode {

T data;

CircLinkNode<T>\* link;

CircLinkNode(CircLinkNode<T>\* ptr = NULL) :link(ptr) {}

CircLinkNode(T d, CircLinkNode<T>\* ptr = NULL) :data(d), link(ptr) {}

};

·单循环链表类的实现

class CircList{

private:

CircLinkNode<T>\* first, \* last;

public:

CircList() : first(nullptr), last(nullptr) {};

CircList(const T& x); // 构造函数

CircList(CircList<T>& L); // 复制构造函数

~CircList(); // 析构函数

int Length()const; // 计算循环链表长度

bool IsEmpty() { return (first == NULL && first == last) ? true : false; }

// 判断空否

CircLinkNode<T>\* getHead()const {return first;};

// 返回附加头结点地址

void setHead(CircLinkNode<T>\* p) { first = p; };

//设置附加头结点地址

CircLinkNode<T>\* search(T x); // 搜索含数据x的元素

CircLinkNode<T>\* Locate(int i); // 搜索第i个元素的地址

T\* getData(int i); // 取出第i个元素的值

void setData(int i, T& x); // 用x修改第i个元素的值

bool Insert(int i, T& x); // 在第i个元素后插入x

bool Remove(int i, T& x); // 删除第i个元素，x返回该元素的值

T FindMinValue(CircList<T>& L); // 寻找链表中的最小值

};

Josephus survivor game.cpp内存储该问题的对应解决方案类：

template <typename Type>

class Josephussolution {

private:

CircList<Type> JosephusList;

int PersonNum;

int StartLocation;

int DeathNum;

int SurvivalPerson;

public:

Josephussolution()

{

PersonNum = 0;

StartLocation = 0;

DeathNum = 0;

SurvivalPerson = 0;

}

void menu();

void Input(int& inputNum, bool opt);

void OutputDeathPerson(int num, int Dead);

void OutputSurvivalPerson();

void JosephusEliminate();

void CreatCircList();

};

## 系统设计

代码被分成多个模块，每个模块负责特定的功能。如CircLinkNode 和 CircList 负责链表的实现，Josephussolution 负责业务逻辑。使用循环链表来模拟人员围成一圈的情况。这种数据结构适合于约瑟夫环问题的需求，因为它允许有效地进行环状遍历和元素删除。

数据和方法被封装在类中，减少了全局状态的使用，这提高了代码的可读性和可维护性。

# 功能设计与项目实现

## 总体设计功能的实现

### 总体系统实现思路

首先明确此系统的主要数据结构：使用循环链表。故无法避免使用到循环链表的各项基本函数。为了提高代码复用效率，创建CircList.h头文件来创建一个循环链表并实现其基本功能。

首先创建循环链表结点结构体并进行初始化，基于此创建循环链表类，实现其构造函数、复制构造函数、析构函数、计算循环链表长度、判断链表是否为空、返回附加头结点地址、设置附加头结点地址、搜索含数据x的元素、搜索第i个元素的地址、取出第i个元素的值、用x修改第i个元素的值、在第i个元素后插入x、删除第i个元素并返回被删除的元素x的值和寻找链表中的最小值。

在链表基本功能完成之后，回归到Josephus问题本身。首先创建该问题的对应解决方案类，里面对应存储存储所有成员序号的循环链表JosephusList、以及总人数(PersonNum)、开始位置(StartLocation)、死亡数字(DeathNum)、幸存者总人数(SurvivalPerson),接着开始初始化菜单界面、输入对应的参数，然后再根据输入的对应参数初始化链表、进行约瑟夫循环。

### 总体系统核心代码

int main() {

Josephussolution<int> JSG;

JSG.menu();

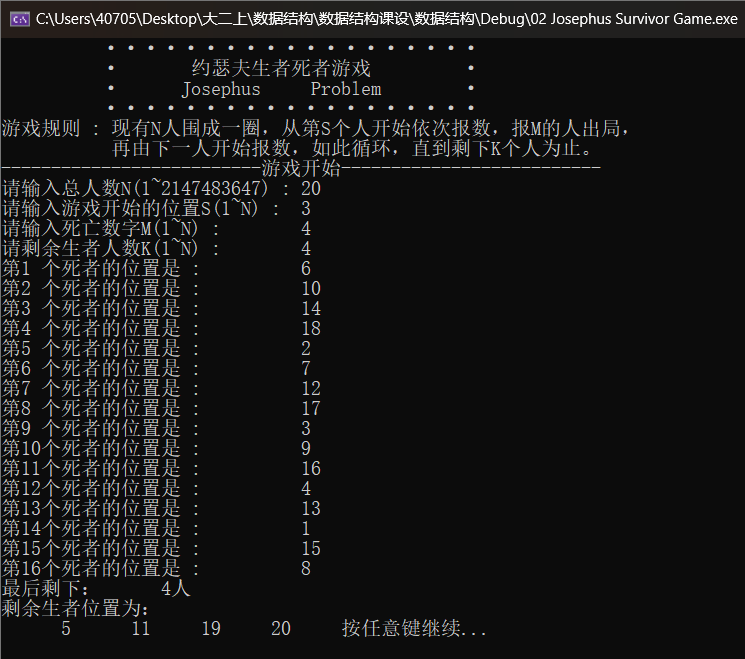
JSG.CreatCircList();

JSG.JosephusEliminate();

return 0;

}

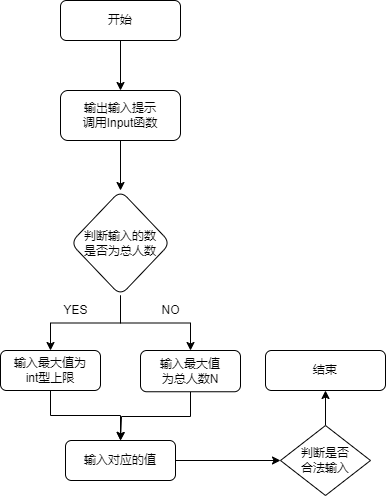
### 总体功能实现示例



## 输入各类数字功能的实现

### 输入功能实现思路

四个参数的输入均有Input函数控制，Input函数的形参作为需要输入的参数的引用，通过bool型变量opt控制输入的最大值（除了第一个人数输入的上限为int型上限，后面三个输入都不能超过最大人数）。通过cin输入参数，如果输入错误或输入的数字不在合法范围内，或者一次性输入多个数据，都会报错。直到读取正确的数据且下一个输入的字符为回车才算输入成功。



### 输入功能核心代码

/\*

\* Function name : Input

\* Function features : Input Relevant Parameters

\* Input parameters : int& inputNum, bool opt

\* Return value : None

\*/

void Input(int& inputNum, bool opt) {

int MaxNum = 0;

if (!opt)

MaxNum = maxPersonNum;

else

MaxNum = PersonNum;

while (1) {

std::cin >> inputNum;

if (std::cin.fail() || inputNum <= 0 || inputNum > MaxNum) {

std::cin.clear();

std::cin.ignore(65536, '\n');

std::cout << "输入数据有误，请重新输入\n";

}

else {

if (std::cin.get() == '\n')

break;

else {

std::cin.clear();

std::cin.ignore(65536, '\n');

std::cout << "输入数据有误，请重新输入\n";

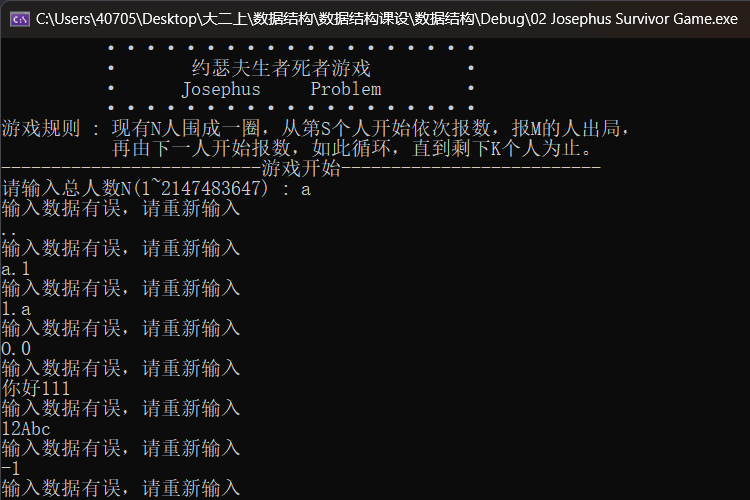
}

}

}

}

### 输入功能实现示例



## 约瑟夫环的核心算法的实现

### 3.3.1 约瑟夫环的核心算法实现思路

首先，算法通过JosephusList.Locate(StartLocation)定位到起始节点p和其前驱节点q。接着，算法进入一个循环，直到被删除的人数达到(PersonNum - SurvivalPerson)。每次循环，算法检查当前的计数num是否符合死亡条件（即(num - 1) % DeathNum == 0）。如果不符合，就简单地移动p和q到下一个节点。如果符合死亡条件，则执行删除操作。首先保存当前的节点to\_delete（即p），然后移动p到下一个节点，并更新q的链接以删除to\_delete。随后调用OutputDeathPerson函数输出被删除节点的信息，并释放该节点的内存。当删除操作完成后，调用OutputSurvivalPerson方法输出最后幸存的人员信息。

### 3.3.2 约瑟夫环的核心算法核心代码

/\*

\* Function name : JosephusEliminate

\* Function features : Handle the Joseph Ring

\* Input parameters : None

\* Return value : None

\*/

void JosephusEliminate()

{

int num = 2;

int count = 1;

CircLinkNode<Type>\* p = JosephusList.Locate(StartLocation);

CircLinkNode<Type>\* q = JosephusList.Locate(StartLocation);

while (q->link != p) {

q = q->link;

}

while (count <= PersonNum - SurvivalPerson)

{

if ((num - 1) % DeathNum != 0 ||num - 1 == 0) {

q = p;

p = p->link;

}

else {

CircLinkNode<Type>\* to\_delete = p;

p = p->link; // 移动 p 到下一个节点

q->link = p; // 删除节点 p

JosephusList.setHead(p);

OutputDeathPerson(count, to\_delete->data);

delete to\_delete; // 释放被删除的节点

count++;

}

num++;

}

OutputSurvivalPerson();

}

### 3.3.3 约瑟夫环的核心算法实现示例

详见下输出生者死者名单实现示例。

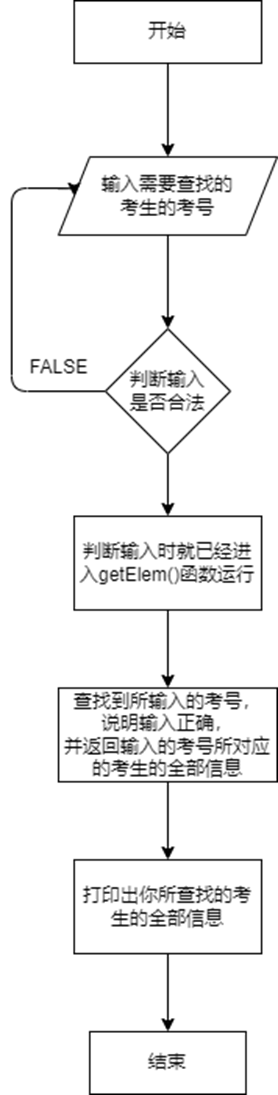
## 输出死者生者名单功能的实现

### 3.4.1 输出死者生者名单功能实现思路

首先输入需要查找的考生的考号(operationNum)。输入及错误处理见3.3.1。

在进行错误处理时，将int getElem(bool InformationorNum)函数的输入置为1，即可将需要查找的考生信息存储到getStudentData中。

对于getElem函数，同样首先新建一个LNode\* p以及int j = 1.遍历链表寻找p->data.examNum == operationNum的位置。若遍历完毕均未找到，就返回错误信息。若找到，就返回考生的序号j或者考生信息getStudentData.二者通过InformationorNum的输入来进行不同的返回。1返回考生信息，0返回序号。



### 3.4.2 输出死者生者名单功能核心代码

/\*

\* Function name : OutputDeathPerson

\* Function features : Output the location of the deceased

\* Input parameters : int num, int Dead

\* Return value : None

\*/

void OutputDeathPerson(int num, int Dead)

{

std::cout << "第" << std::setw(2) << num;

std::cout << std::setw(26) << std::left << "个死者的位置是 : ";

std::cout << Dead << "\n";

}

/\*

\* Function name : OutputSurvivalPerson

\* Function features : Output the location of the living

\* Input parameters : None

\* Return value : None

\*/

void OutputSurvivalPerson()

{

int cnt = 0;

int MinValue = JosephusList.FindMinValue(JosephusList);

CircLinkNode<Type>\* p;

std::cout << std::setw(16) << std::left << "最后剩下：" << SurvivalPerson << "人\n";

std::cout << "剩余生者位置为：\n ";

if (!JosephusList.IsEmpty()) {

if (JosephusList.Length() == 1)

p = JosephusList.getHead();

else

p = JosephusList.search(MinValue);

while (cnt < SurvivalPerson) {

std::cout << std::setw(7) << p->data;

cnt++;

p = p->link;

if (cnt % 7 == 0)

std::cout << "\n ";

}

}

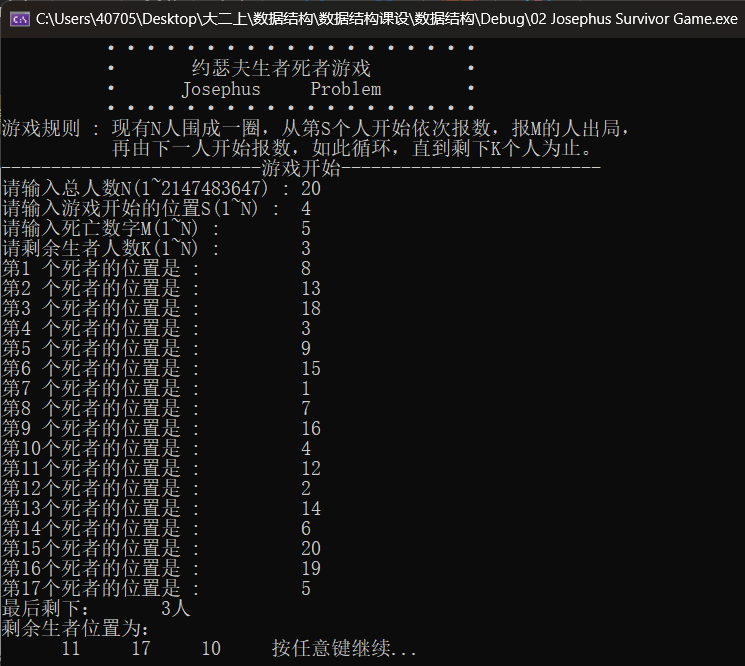
else {

std::cout << "链表为空，没有生者。\n";

}

}

### 3.4.3 输出死者生者名单功能实现示例



## 本项目涉及CircList.h内容的主要实现

### 3.5.1 节点结构体和循环链表类的定义

定义了循环链表的节点结构体CircLinkNode，包含数据域data和指向下一个节点的指针link。初始化一个空链表或以给定值创建链表的头节点。用于复制一个现有的链表（在提供的代码中此函数体为空，可能需要实现）。释放链表中的所有节点，防止内存泄漏。

### 3.5.2 CircList.h核心代码

template <class T>

struct CircLinkNode {

T data;

CircLinkNode<T>\* link;

CircLinkNode(CircLinkNode<T>\* ptr = NULL) :link(ptr) {}

CircLinkNode(T d, CircLinkNode<T>\* ptr = NULL) :data(d), link(ptr) {}

};

template <class T>

class CircList{

private:

CircLinkNode<T>\* first, \* last;

public:

CircList() : first(nullptr), last(nullptr) {};

CircList(const T& x); // 构造函数

CircList(CircList<T>& L); // 复制构造函数

~CircList(); // 析构函数

int Length()const; // 计算循环链表长度

bool IsEmpty() { return (first == NULL && first == last) ? true : false; }

// 判断空否

CircLinkNode<T>\* getHead()const {return first;};

// 返回附加头结点地址

void setHead(CircLinkNode<T>\* p) { first = p; };

//设置附加头结点地址

CircLinkNode<T>\* search(T x); // 搜索含数据x的元素

CircLinkNode<T>\* Locate(int i); // 搜索第i个元素的地址

T\* getData(int i); // 取出第i个元素的值

void setData(int i, T& x); // 用x修改第i个元素的值

bool Insert(int i, T& x); // 在第i个元素后插入x

bool Remove(int i, T& x); // 删除第i个元素，x返回该元素的值

T FindMinValue(CircList<T>& L); // 寻找链表中的最小值

};

# 项目测试

## 功能测试

### 基本功能测试

**测试用例：**10 3 1 2；7 2 3 3；6 5 1 1；8 3 2 2；5 2 1 2；5 5 1 1

**用例说明：**这几项测试分别为标准情况测试、非首位测试、大间隔测试、多次循环测试、较小群体测试和间隔等于人数测试。

**预期结果：**程序应该按照规定间隔淘汰人员，最终留下两名指定的生还者。

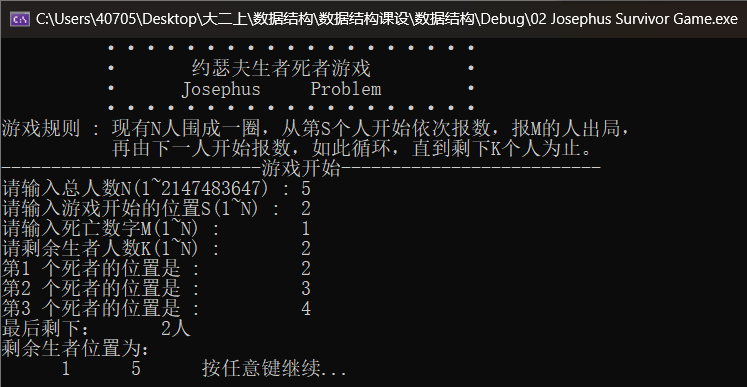
**实验结果：**

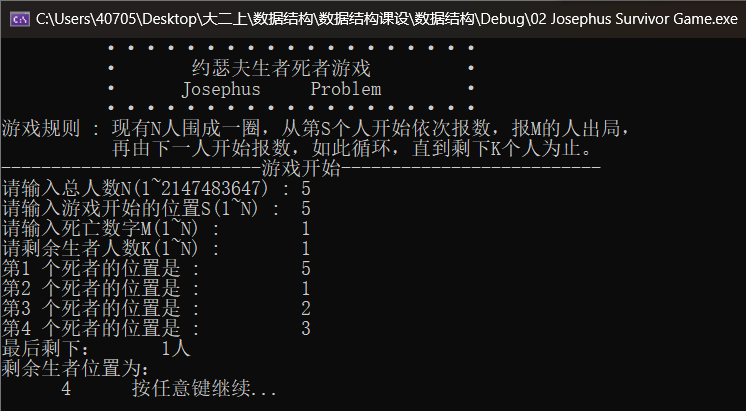












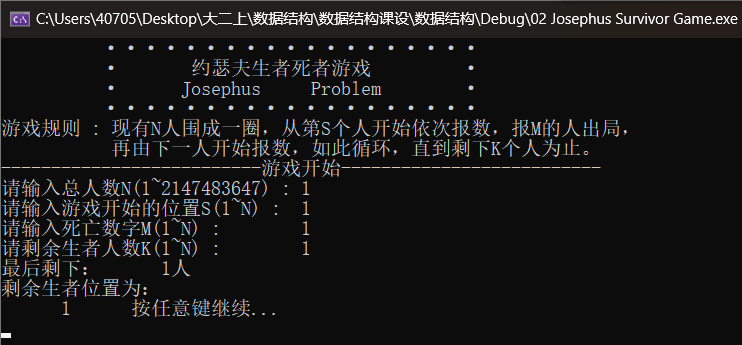
### 4.1.2 边界条件测试

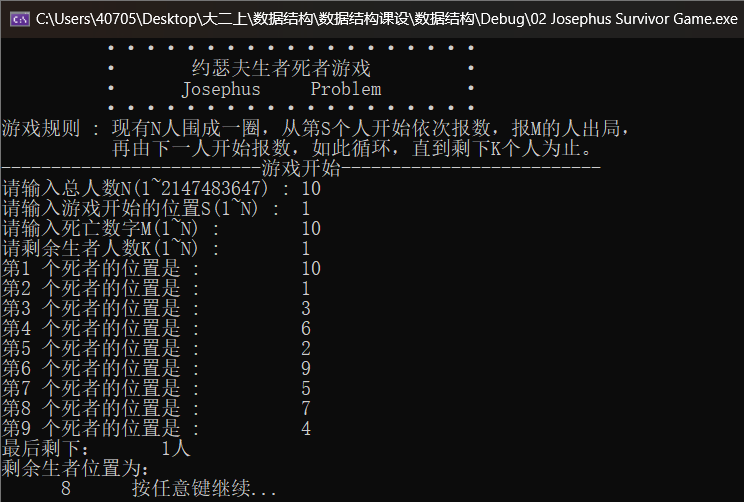
**测试用例：**1 1 1 1；10 1 10 1；10 2 1 10

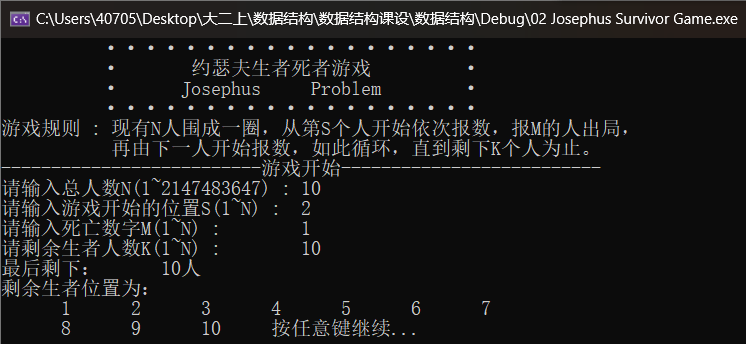
**用例说明：**这几项测试分别为最小值测试、边界值测试、所有人生还测试。

**预期结果：**程序应该在极端情况下也能正确运行，按照规定间隔淘汰人员，最终留下两名指定的生还者。

**实验结果：**







## 错误测试

### 输入验证错误

#### 4.2.1.1 输入验证错误判断思路

输入非法值（如负数、非数字字符、超出范围的数字）来测试程序对错误输入的处理能力。检查程序是否能妥善处理并提示用户重新输入。

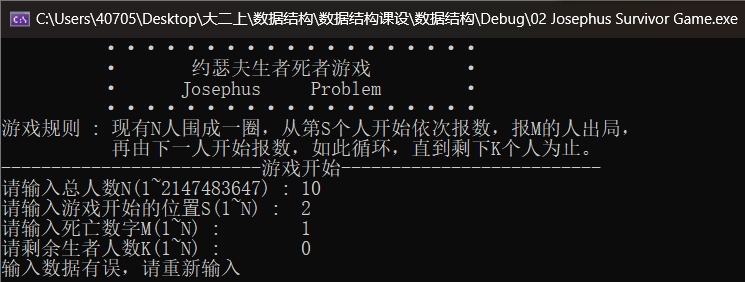
#### 4.2.1.3输入示例

**测试用例：**10 2 1 0；

**用例说明：**没有生还者；

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

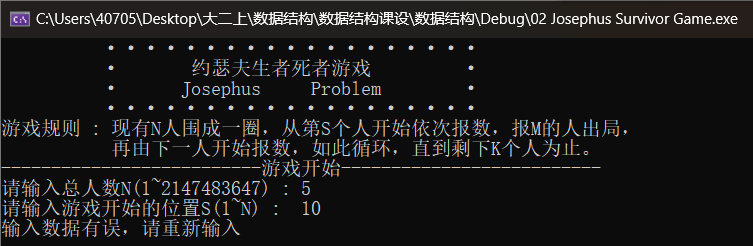


**测试用例：**5 10 1 1；

**用例说明：**测试开始的位置大于总人数位置；

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

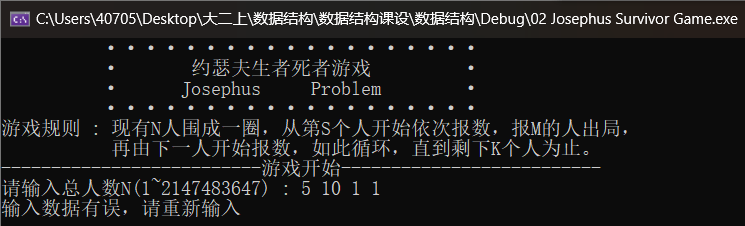


**测试用例：**5 10 1 1；

**用例说明：**测试多个输入

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

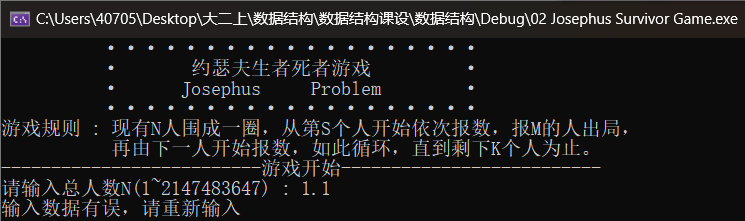


**测试用例：**1.1；

**用例说明：**测试小数；

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**



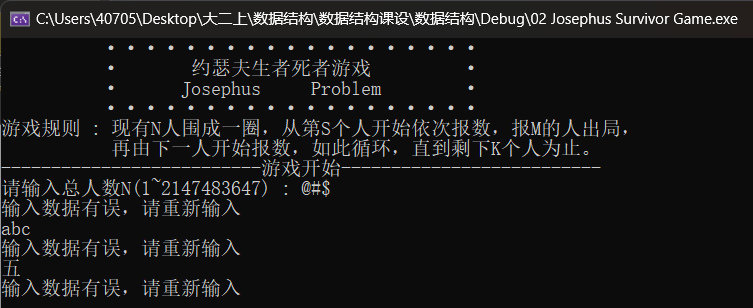
### 异常情况处理

**测试用例：**abc/ @#￥ / 五

**用例说明：**测试非法输入（如字母）

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

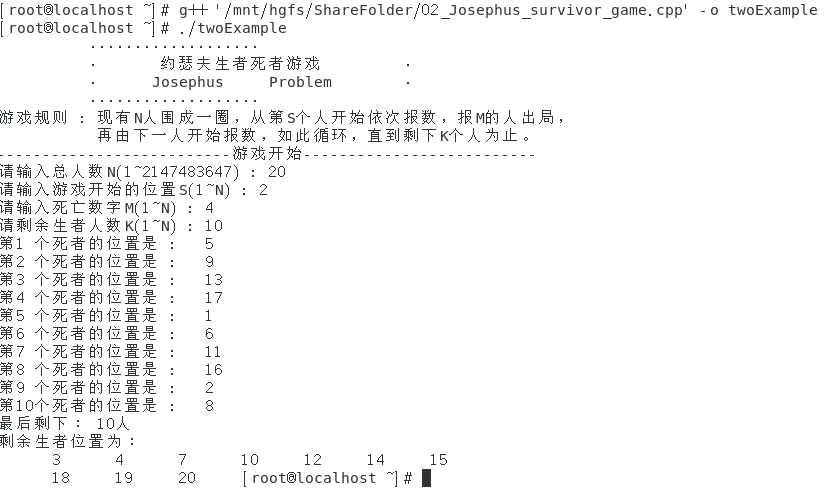


## Linux环境测试

**编译命令：** g++ '/mnt/hgfs/ShareFolder/02\_Josephus\_survivor\_game.cpp' -o twoExample

**运行命令：** ./twoExample

**实验结果：**



1. **集成开发环境与编译运行环境**

Windows系统：Windows 11 x64

Windows集成开发环境：Microsoft Visual Studio 2022 (Release模式)

Windows编译运行环境：本项目适用于x86架构和x64架构

Linux系统：CentOS 7 x64