**项目说明文档**

**数据结构课程设计**

**——排序算法比较**

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 十一 月 二十 日

目录

1 项目分析 1

1.1 项目背景分析 1

1.2 项目功能分析 1

2 项目设计 1

2.1 数据结构设计 1

2.2 类结构设计 2

2.3 成员与操作设计 2

2.4 系统设计 3

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 4

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 5

3.2.1 输入功能实现思路 5

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 7

3.3.1 约瑟夫环的核心算法实现思路 7

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 8

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 11

3.5 本项目涉及CircList.h内容的主要实现 11

3.5.1 节点结构体和循环链表类的定义 11

3.5.2 CircList.h核心代码 11

4 项目测试 12

4.1 功能测试 12

4.1.1 基本功能测试 12

4.1.2 边界条件测试 15

4.2 错误测试 16

4.2.1 输入验证错误 16

4.2.1.1 输入验证错误判断思路 16

4.2.1.3输入示例 16

4.2.2 异常情况处理 18

4.3 Linux环境测试 18

5 集成开发环境与编译运行环境 19

# 项目分析

## 项目背景分析

该项目旨在通过实现不同排序算法，比较它们在排序一组随机生成的数值数据时的性能，包括排序时间和交换次数。排序算法在计算机科学中起着至关重要的作用。不同的排序算法具有不同的性能特征，如时间复杂度和空间复杂度。这个项目旨在比较一些常见的排序算法，包括快速排序、直接插入排序、冒泡排序和选择排序。

项目不仅对数据进行排序，还会统计每种排序方法所花费的排序时间和交换次数。这些性能指标可以帮助用户评估不同排序算法的效率和适用性。

## 项目功能分析

用户使用随机函数生成指定数量的随机数，并存储在randNum数组中。项目涵盖了四种排序算法：快速排序、直接插入排序、冒泡排序和选择排序。每种排序算法的功能是将随机生成的随机数数组（randNum）按升序排序，并将排序后的结果存储在sortNum数组中。各排序算法的功能分别在对应的函数中实现。

项目能够统计每种排序算法的性能指标，包括排序时间和交换次数。排序时间使用高精度计时器（LARGE\_INTEGER）来测量，以秒为单位。交换次数在排序过程中计数，以帮助评估排序算法的效率。

# 项目设计

## 数据结构设计

项目中使用了多个数组来存储随机数和排序后的结果。项目中定义了一个名为SortingAlgorithms的模板类，用于封装排序算法和相关功能。这个类包括了成员变量和成员函数，用于存储和操作数据，实现不同的排序算法，以及与用户交互。

SortingAlgorithms类中的成员变量包括randNum（用于存储随机数）、sortNum（用于存储排序后的数据）、chooseNum（用于用户选择排序算法）、maxNum（用户定义的随机数个数）、sortCnt（用于统计交换次数）等。SortingAlgorithms类中的成员函数包括了各种排序算法的实现（如bubbleSort、selectionSort等），输入随机数的函数（inputNum），生成随机数的函数（generateRandomNumbers），选择排序算法和性能统计的函数（chooseAlgorithms），以及其他辅助函数（如print、Exit等）。

## 类结构设计

在项目中，排序算法的结构设计主要体现在SortingAlgorithms类中。这个类封装了多种排序算法以及与排序相关的功能。randNum：私有数组，用于存储随机生成的随机数。sortNum：私有数组，用于存储排序后的数据。chooseNum：整数，用于记录用户选择的排序算法。maxNum：整数，用户定义的随机数个数。sortCnt：整数，用于统计排序时的交换次数。

inputNum()：输入数字的函数，用于获取用户输入的整数，并进行验证。generateRandomNumbers()：生成随机数的函数，使用随机函数生成指定数量的随机数，并存储在randNum数组中。initSort()：将sortNum数组初始化为randNum数组，以便排序。chooseAlgorithms()：排序算法选择函数，根据用户选择的算法，执行相应的排序算法，并记录性能指标。各种排序算法函数（如bubbleSort、selectionSort等）：用于实现不同的排序算法。getMax()：获取sortNum数组中的最大值。countSort()：计数排序算法，用于基数排序。radixSort()：基数排序算法。print()：打印数组函数，用于显示排序后的结果。Exit()：退出程序函数，用于等待用户按键并清除输入缓冲区。

## 成员与操作设计

template <typename Type>

class SortingAlgorithms {

private:

Type randNum[MAX\_SIZE];

Type sortNum[MAX\_SIZE];

public:

int chooseNum = -1;

int maxNum = -1;

int sortCnt = 0;

void inputNum(int& num,bool opt);

void generateRandomNumbers();

void chooseAlgorithms();

void bubbleSort(); //冒泡排序

void selectionSort(); //选择排序

void directInsertionSort(); //直接插入排序

void ShellSort(); //希尔排序

void quickSort(int left, int right); //快速排序

void heapify(int n, int i); //调整堆

void heapSort(); //堆排序

void merge(int left, int mid, int right); //合并有序数组

void mergeSort(int l,int r); //归并排序

// 获取最大值

int getMax() {

int mx = sortNum[0];

for (int i = 1; i < maxNum; i++)

if (sortNum[i] > mx)

mx = sortNum[i];

return mx;

}

void countSort(int exp);

void radixSort(); //基数排序

// 打印数组函数

void print() {

std::cout << "\n";

for (int i = 0; i < maxNum; i++){

std::cout << sortNum[i] << " ";

if (i % 9 == 0 && i != 0)

std::cout << "\n";

}

}

void initSort();

void Exit() {

std::cout << "按任意键继续...\n";

// 清除输入缓冲区

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

// 等待用户按键

std::cin.get();

}

};

## 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();

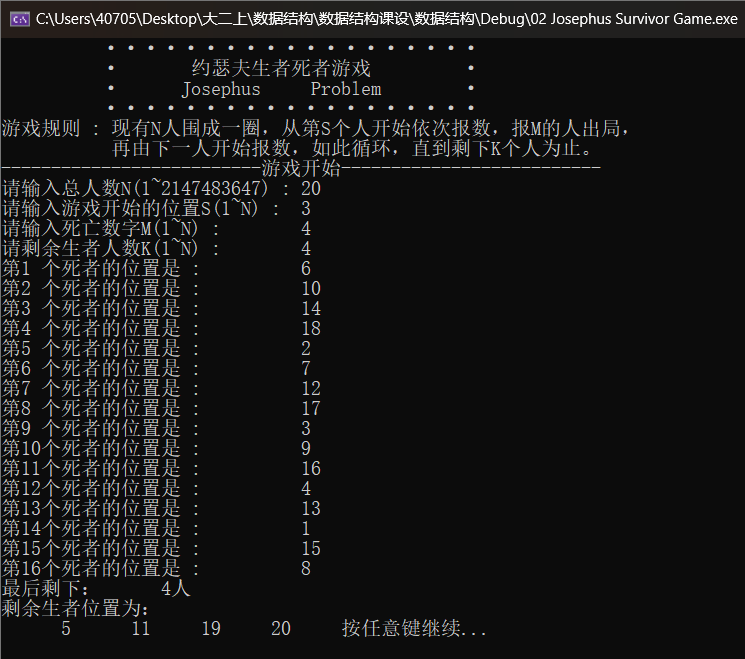
JSG.CreatCircList();

JSG.JosephusEliminate();

return 0;

}

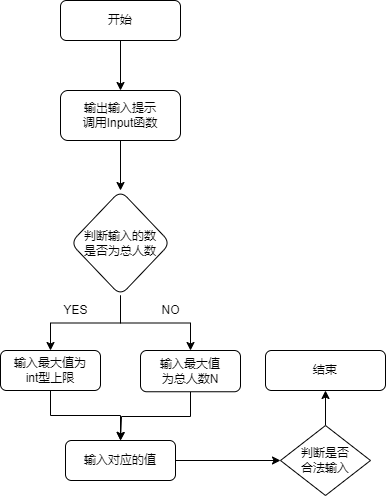
### 总体功能实现示例



## 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";

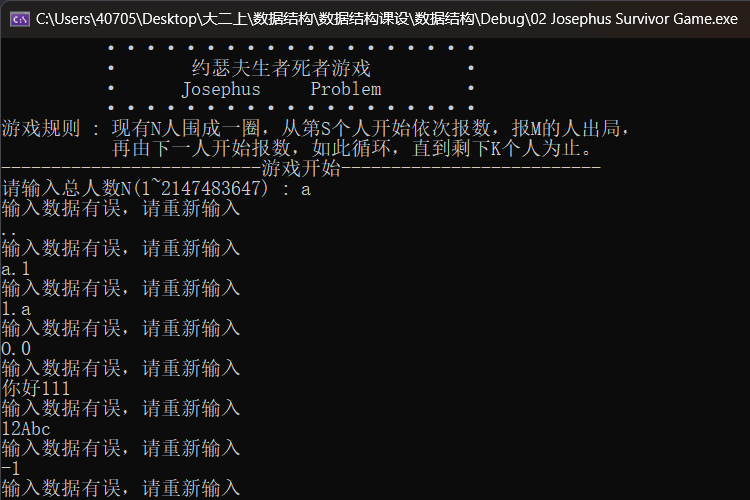
}

}

}

}

### 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。

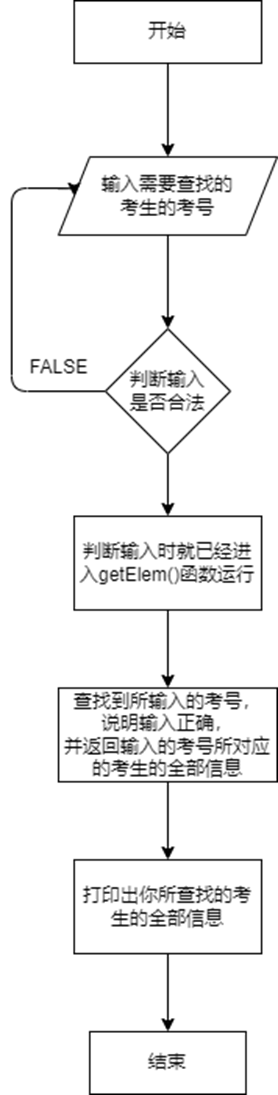
## 输出死者生者名单功能的实现

### 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 ";

}

}

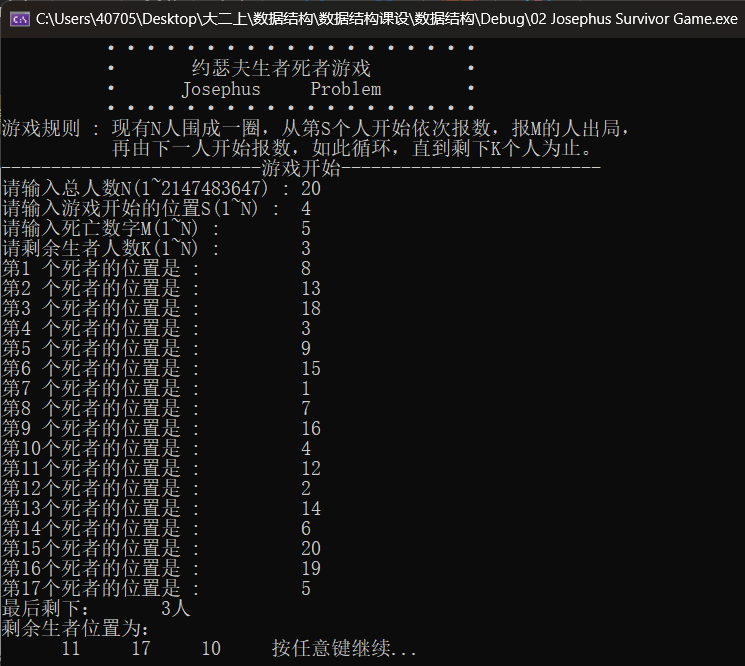
else {

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。

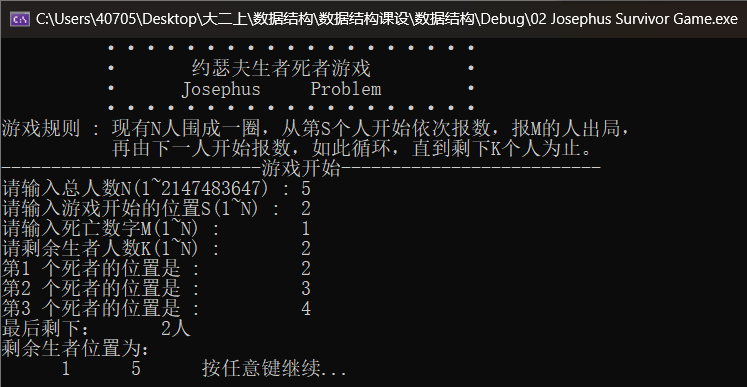
**实验结果：**

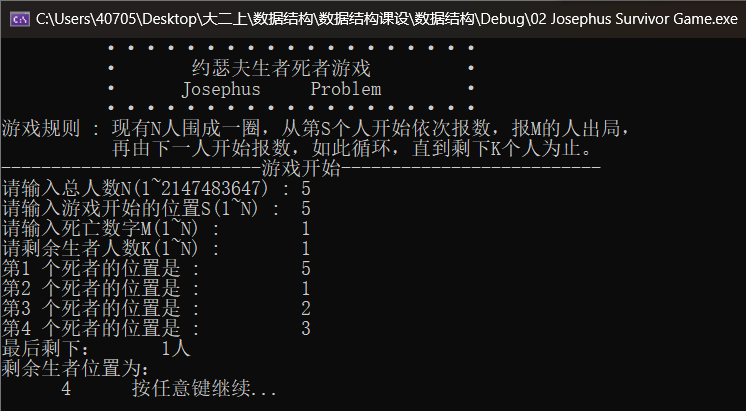












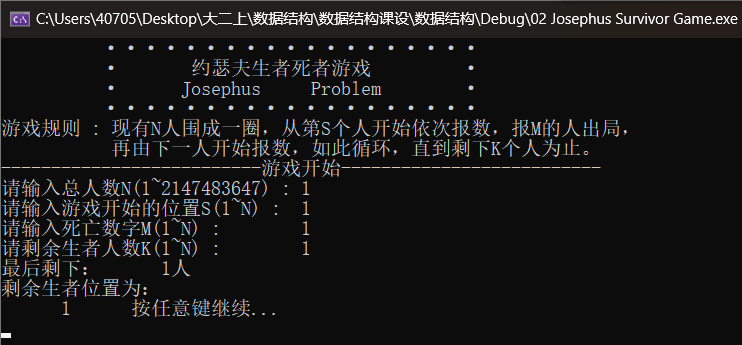
### 4.1.2 边界条件测试

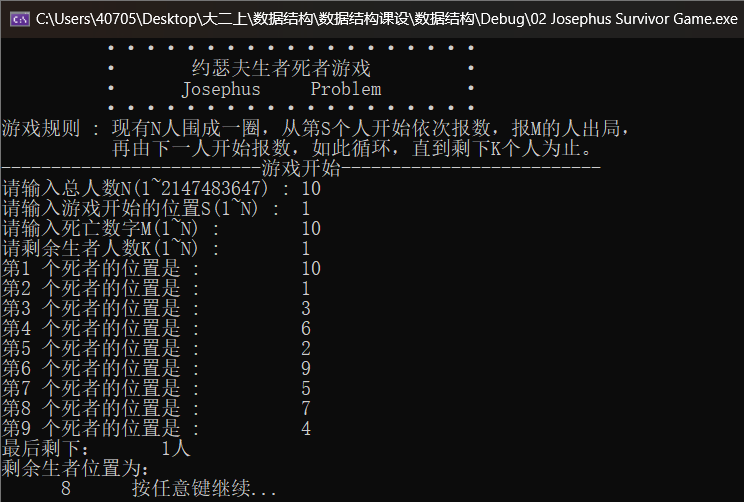
**测试用例：**1 1 1 1；10 1 10 1；10 2 1 10

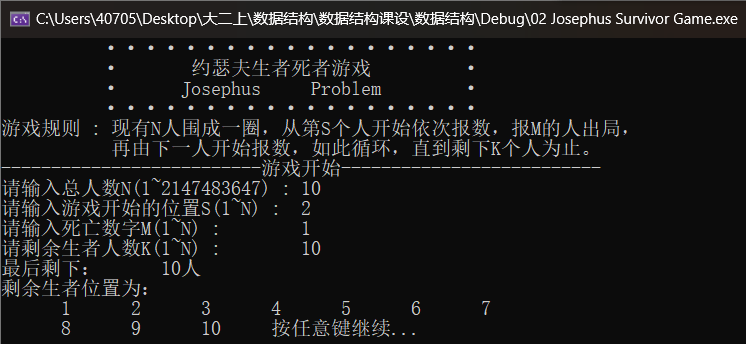
**用例说明：**这几项测试分别为最小值测试、边界值测试、所有人生还测试。

**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。

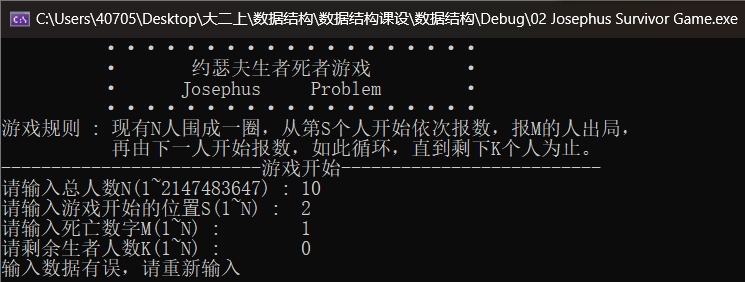
#### 4.2.1.3输入示例

**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：**

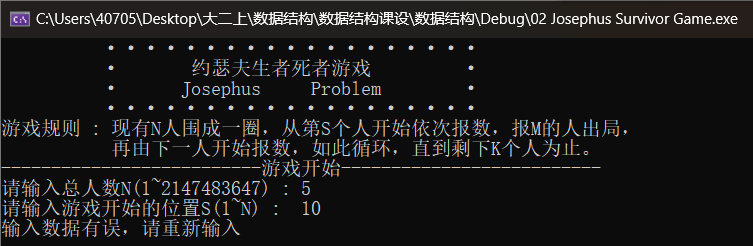


**测试用例：**5 10 1 1；

**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：**

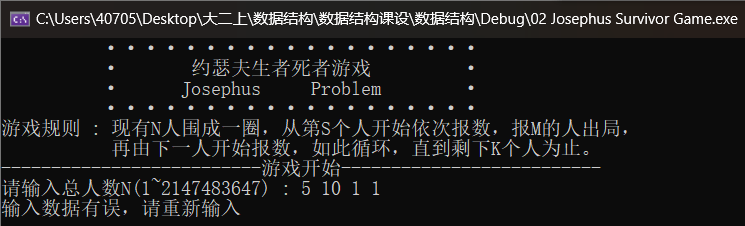


**测试用例：**5 10 1 1；

**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：**

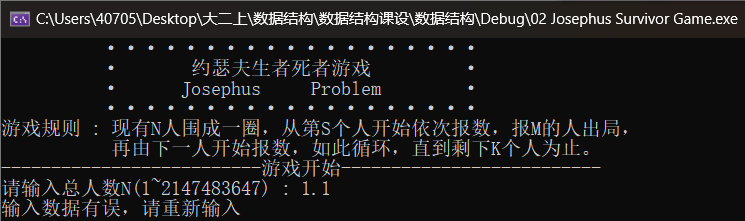


**测试用例：**1.1；

**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：**



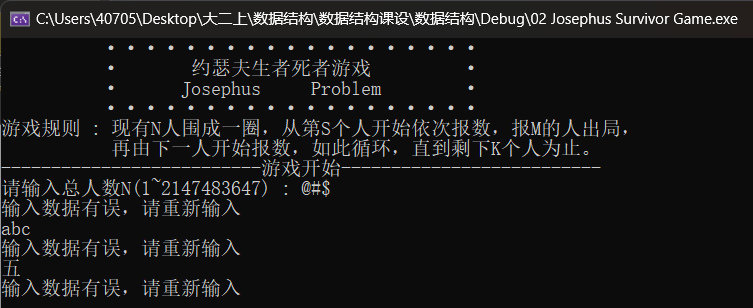
### 异常情况处理

**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：**

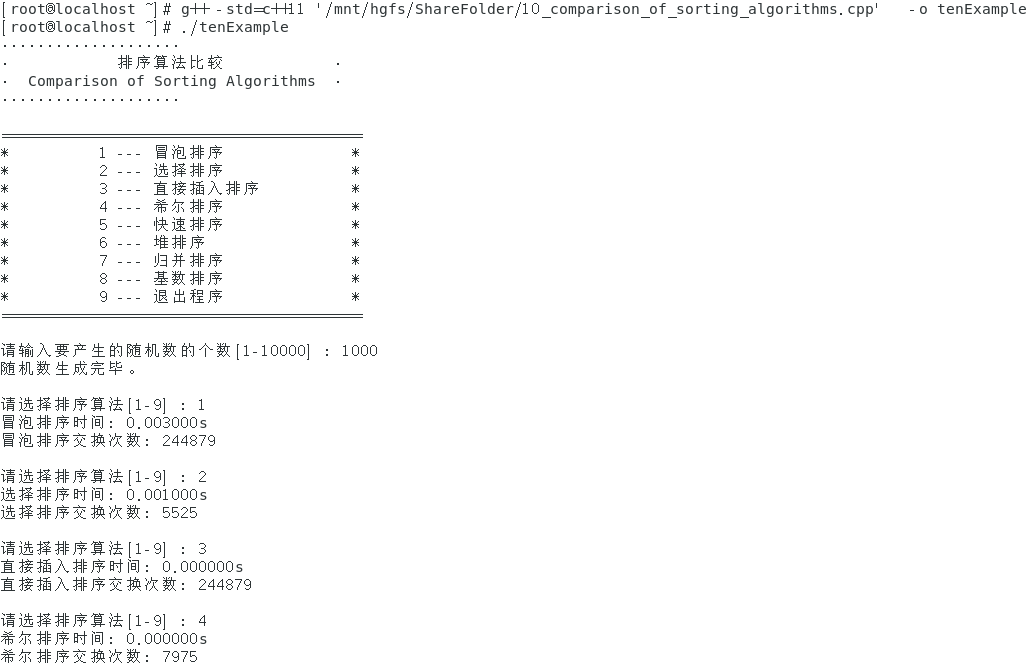


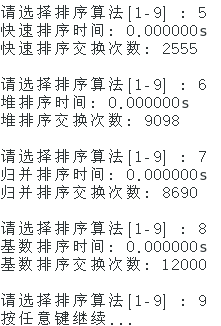
## Linux环境测试

**编译命令：** g++ -std=c++11 '/mnt/hgfs/ShareFolder/ 10\_comparison\_of\_sorting\_algorithms' -o tenExample

**运行命令：** ./tenExample

**实验结果：**





# 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