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： 曹晓慈

学 号： 2052844

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 分析

### 背景分析

据说著名犹太历史学家 Josephus有过以下的故事：在罗马人占领乔塔帕特后，39 个犹太人与Josephus及他的朋友躲到一个洞中，39个犹太人决定宁愿死也不要被敌人抓到，于是决定了一个自杀方式，41个人排成一个圆圈，由第1个人开始报数，每报数到第3人该人就必须自杀，然后再由下一个重新报数，直到所有人都自杀身亡为止。然而Josephus 和他的朋友并不想遵从，Josephus要他的朋友先假装遵从，问他俩站在那两个位置才能活着留下。并打印出死亡位置

### 2.功能分析

约瑟夫游戏的大意：30 个游客同乘一条船，因为严重超载，加上风浪大作，危险万分。因此船长告诉乘客，只有将全船一半的旅客投入海中，其余人才能幸免于难。无奈，大家只得同意这种办法，并议定30 个人围成一圈，由第一个人数起，依次报数，数到第9 人，便把他投入大海中，然后再从他的下一个人数起，数到第9 人，再将他投入大海中，如此循环地进行，直到剩下15 个游客为止。问：哪些位置是将被扔下大海的位置？

故本问题旨在写出一个算法来确定最终剩下的人。

# 设计

1. 数据结构设计

根据上面的题意可知该系统要求大量删除操作，而链表进行删除操作十分简便，因此考虑使用链表数据结构。同时，易知该题目需要尾结点连接到头结点，故采用循环链表的数据结构。

1. 类结构设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（LNode）与链表类（LinkList），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。为方便处理，本系统采用struct描述链表结点类（Node），这样使得链表结点类（CircleList）可以访问链表结点。

1. 成员与操作设计

链表结点类：

class Node

{

public:

Node\* next;

int data;

};

链表类：

class CircleList

{

private:

Node\* head;

int length;

public:

CircleList()

{

head = new Node();

head->next = head;

head->data = 0;

length = 0;

}

~CircleList()

{

delete(head);

}

void createCircleList(int n); //创建单向循环链表

void traverseNode(); //遍历链表

//void deleteNode(int n); //删除位置为n的结点

//void insertNode(int n, int data);//在制定位置插入结点

//Node\* Locate(int i); //查找位置i的节点

Node\* getHead(); //获得头指针

void Length();

int getLength(); //得到链表长度

bool isEmpty(); //判断链表是否为空

};

## 实现

1. 创建循环链表

void CircleList::createCircleList(int n) //创建的最后将尾结点指向头结点

{

if (n < 0)

{

cout << "你输入的长度不正确 " << endl;

}

else

{

int m = 0;

length = n;

Node\* p, \* q;

p = head;

while (m != n)

{

m++;

q = new Node();

q->data = m;

p->next = q;

q->next = head;

p = q;

}

}

}

1. 查找循环链表第i个元素并返回其相应的地址

当i小于0时直接返回NULL，否则则从头部开始遍历的同时让k从0开始自增直到k=i即为所找元素，返回其地址。

Node\* CircleList::Locate(int i)

{

if (i < 0)

return NULL;

Node\* current = head;

int k = 0;

while (current != NULL && k < i)

{

current = current->next;

k++;

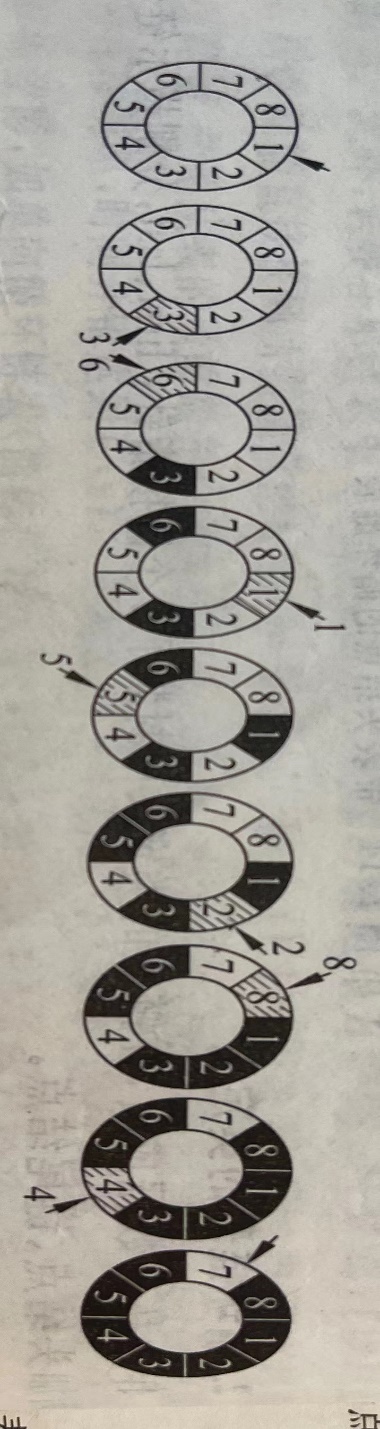
}

return current;

}

3.约瑟夫生死环算法实现

容易知道该算法要用到循环链表，以及死亡人数N-K，死亡数字M，以及开始人S，从S所在人开始点数循环，每点到数字M就将该结点删去并从下一结点开始重新报数，循环直到死亡的人数为N-K人时结束循环。具体在实现过程中还出现了头结点的指针出现错误等等，对于此进行单独的处理，让头结点指针始终指向链表存在的第一个元素。以下是代码实现：



void Josephus(CircleList& Js, int N, int M, int S, int K)

{

Node\* p = Js.Locate(S), \* pre = NULL;

Node\* first = Js.getHead();

int i, j;

for (i = 1; i <= N - K; i++) //外圈循环计算死亡人数

{

for (j = 1; j < M; j++) //内圈循环进行报数的记录

{

pre = p;

p = p->next;

if (p == first) //跳过头结点

{

p = p->next;

}

}

cout << "第" << i << "个死者的位置是： " << p->data << endl;

pre->next = p->next;

if (first->next == p) //保证头结点的next指针指向还存在的节点

first->next = p->next;

delete p;

p = NULL;

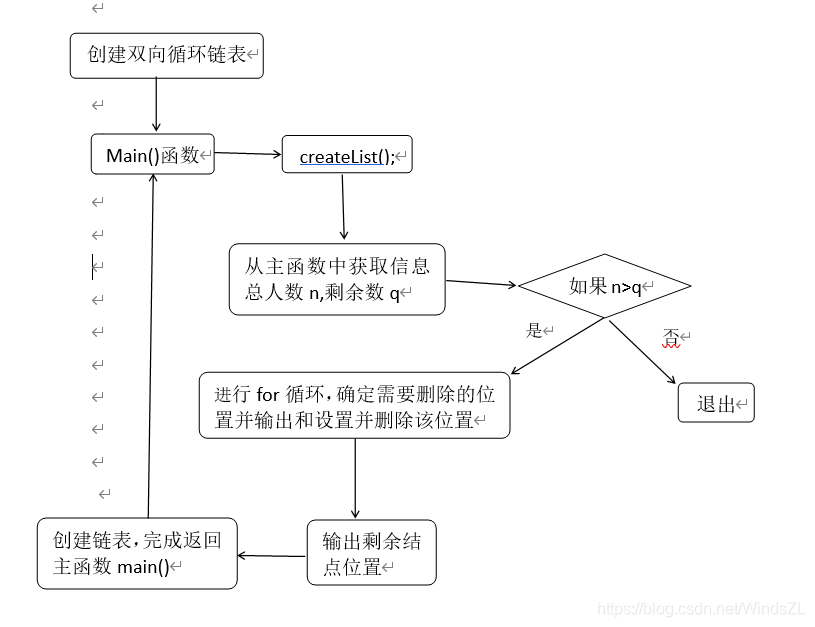
p = pre->next;

if (p == first) //计数跳过头结点

p = p->next;

}

}

1. 总体系统实现
2. 

创建单向链表

int main()

{

CircleList list;

int N, S, M, K;

人数N输入；

list.createCircleList(N);

S，M,K的输入；

Josephus(list, N, M, S, K);

list.Length(); //更新链表长度

输出提示

list.traverseNode(); //遍历输出

return 0;

}

## 四.测试

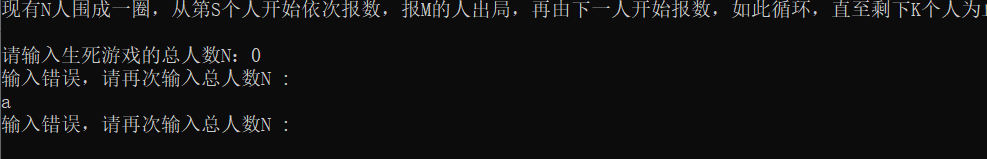
### 4.1边界测试

#### 总人数输入错误

**测试用例：**输入总人数为非正数或不为int型

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

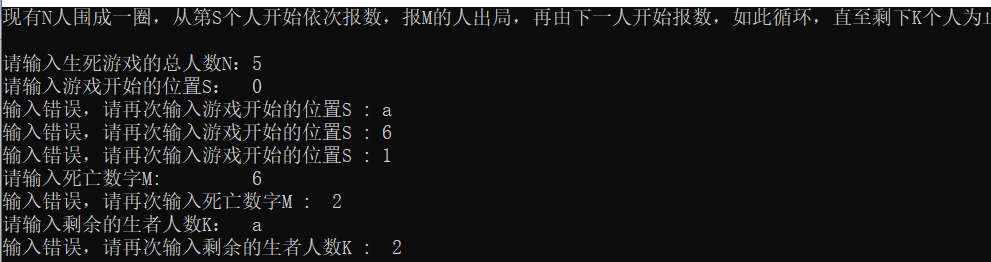


#### 游戏开始位置，死亡数字和幸存人数输入错误

**测试用例：**输入总人数为非正数或不为int型或大于总人数N

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

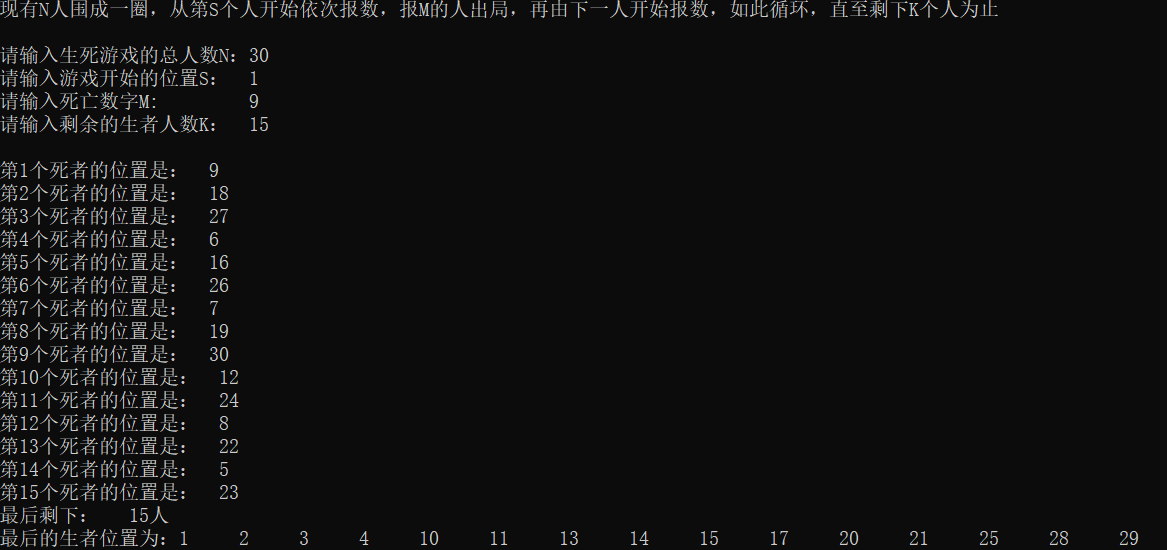


### 4.2 数据测试

测试用例：30 1 9 15

预期结果：程序正常运行且给出正确答案

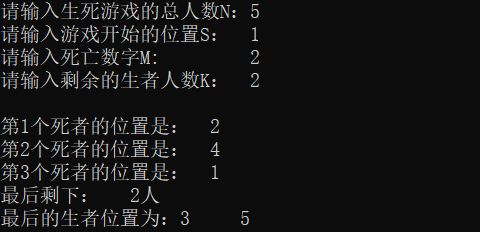
实验结果：



测试用例：5 1 2 2

预期结果：程序正常运行且给出正确答案

实验结果：



测试用例：10 1 3 1

预期结果：程序正常运行且给出正确答案

实验结果：

