目录

1.项目背景

本项目基于约瑟夫生死者游戏，约瑟夫生者死者游戏的大意是：30个旅客同乘一条船，因为严重超载，加上风高浪大危险万分；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，大家只得统一这种方法，并议定30个人围成一圈，由第一个人开始，依次报数，数到第9人，便将他投入大海中，然后从他的下一个人数起，数到第9人，再将他投入大海，如此循环，直到剩下15个乘客为止。问哪些位置是将被扔下大海的位置。

本游戏的数学建模如下：假如N个旅客排成一个环形，依次顺序编号1, 2, ..., N。从某个指定的第S号开始。沿环计数，每数到第M个人就让器出列，且从下一个人开始重新计数，继续进行下去。这个过程一直进行到剩下K个旅客为止。

本游戏用户输入的内容包括：

1. 旅客的个数，也就是N的值
2. 离开旅客的间隔书，也就是M的值
3. 所有旅客的序号作为一组数据要求存放在某种数据结构中

随后程序将输出：

1. 离开旅客的序号
2. 剩余旅客的序号

2.需求分析

本项目为实现约瑟夫生死者的游戏模拟，接受用户输入的游戏参数后，通过程序得出离开旅客的序号和剩余旅客的序号

3.设计思路

数据结构

约瑟夫生死者游戏的特点就是所有人的序列是一个循环序列，每次计数若超过最后一个人之后即跳转至第一个人，这样的特点选择单向循环链表作为数据结构是非常合适的，将每一位旅客作为一个链表节点

功能实现

采用带头结点的单向循环链表有序存储下所有旅客，用户数据输入完毕后，将链表指针的位置调整至游戏的起始位置，随后开始循环计数，每一次计数到达后，输出该节点旅客信息，随后执行删除节点操作，当删除的旅客数达到要求后，从头结点开始遍历链表，输出所有剩余旅客信息

4.核心代码说明

节点类

```
struct Node {
    int postion;
    Node *pNext;
    Node(int n):postion(n),pNext(NULL){}
    Node():postion(0),pNext(NULL){}
};
```

用于存储单个旅客信息，包括位置（position），下个节点指针（*pNext)

包含两个构造函数：默认构造函数，带位置的构造函数

链表类

```
//Create a circular link list to stimulate Joseph circle
class CircularLinkList {
private:
    Node *head;
    Node *tail;
    int length;
public:
    //Default constructor
    CircularLinkList() {
        head = new Node(0);
    }
    //Overload constructor
    CircularLinkList(int amount);
    //Delete node
    void DeleteNode(Node *node);
    //Remove people in terms of start and gap
    void JosephStart(int start, int gap, int survival);
};
```

包含三个私有成员变量：头结点指针（*head），尾节点指针（*tail），长度（length）

成员函数的功能见注释

约瑟夫环函数

```
//Remove people in terms of start and gap
void JosephStart(int start, int gap, int survival) {
  Node* temp = head;
  //Adjust the pointer to start position
  for (int i = 1; i <= start; i++) {
    temp = temp->pNext;
  }
  while (length > survival) {//Keep recycling until the length less than that user entered
    int k = 1;
    //Adjust to the position one before required
    for (int i = 0; i < gap-1; i++) {
      temp = temp->pNext;
    }
    //Save the pointer
    Node* delNode = temp;
    temp = temp->pNext;
    //Output information of the node
    cout << "第" << k << "个死者的位置时: " << "\t" << delNode->postion << endl;
    //Delete the node
    DeleteNode(delNode);
    k++;
  }
  //Output the survivals' information
  cout << endl << "最后剩下" << survival << "人" << endl;
  cout << "剩余生者的位置为: ";
  Node* show = head->pNext;
  for (int i = 0; i < survival; i++) {
    cout << "\t" << show->postion;
    show = show->pNext;
  }
  return;
}
```

首先调整指针位置到起始节点，然后在while循环中进行计数，一次到达要删除的节点，while循环终止的条件为剩余人数达到要求，在每一次遍历至待删除节点时，先输出其位置信息，随后调用`DeleteNode`函数删除该节点

```
//Delete node
void DeleteNode(Node *node) {
  Node* temp = head;
  while (temp->pNext->postion != node->postion && temp!= tail) {
    temp = temp->pNext;
  }
  if (temp != tail) {
    Node *delNode = temp->pNext;
    temp->pNext = temp->pNext->pNext;
    delete delNode;
    length--;
  }
  else{
    cout<<"Can't' find the node!"<<endl;
```

```
    }
    return;
}
```

因为传入的`node`是待删除节点的前一个节点的指针，所以在该函数中需要重新遍历来找到待删除节点位置，然后删除

函数接口说明

返回值类型	成员函数名	参数	属性	功能
\	CircularLinkList	\	public	默认构造函数
\	CircularLinkLIst	(int amount)	public	构造函数
void	DeleteNode	(Node *node)	public	删除节点
void	JosephStart	(int start,int gap,int survival)	public	约瑟夫环

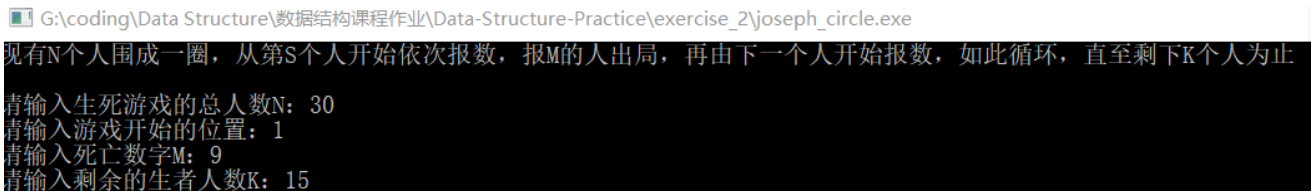
5.使用方法及函数动能那个演示

双击`joseph_circle.exe`运行程序

输入数据

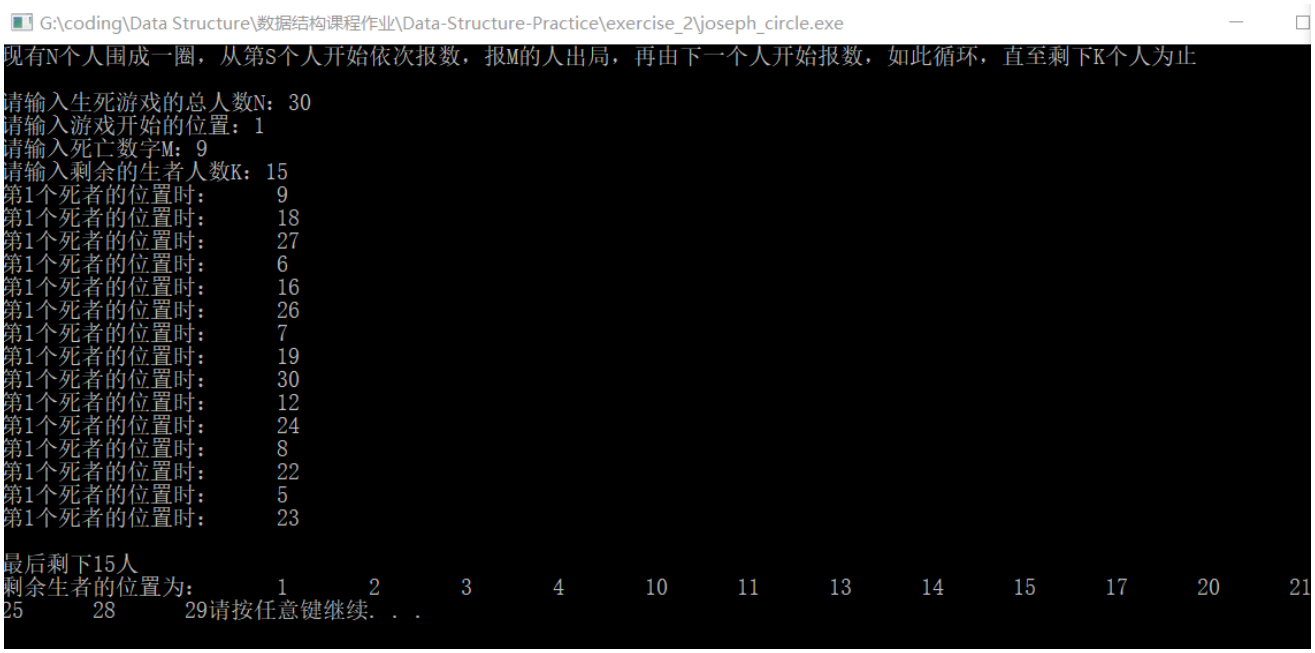
根据程序提示依次输入：

- 游戏总人数N
- 游戏开始的位置S
- 死亡数字M
- 剩余的生者人数K



输出数据

输入完成后，程序将一次输出每一个死者位置，直至剩余人数达到用户输入人数，最后输出所有生者位置，游戏结束



按任意键退出程序，需要再次模拟则重复上述步骤