# 1 项目简介

本游戏的规则如下：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个人就让其出列，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。

## 本程序就是对这一过程的模拟，K、M、N、S均由用户输入，最终按顺序输出离开旅客的序号以及留下的旅客的序号。

# 2 程序说明

## 2.1 数据结构设计

考虑到约瑟夫生死环游戏的流程和单循环链表这一数据结构非常类似，因此，采用单循环链表来模拟约瑟夫生死环的游戏流程。游戏中淘汰某序号的旅客在单循环链表中即根据遍历的位置删除某个节点，当删除的节点数量达到用户预设的数量之后，遍历链表，输出旅客信息即可，这样一来就可以利用单循环链表完成对约瑟夫生死环游戏的模拟。

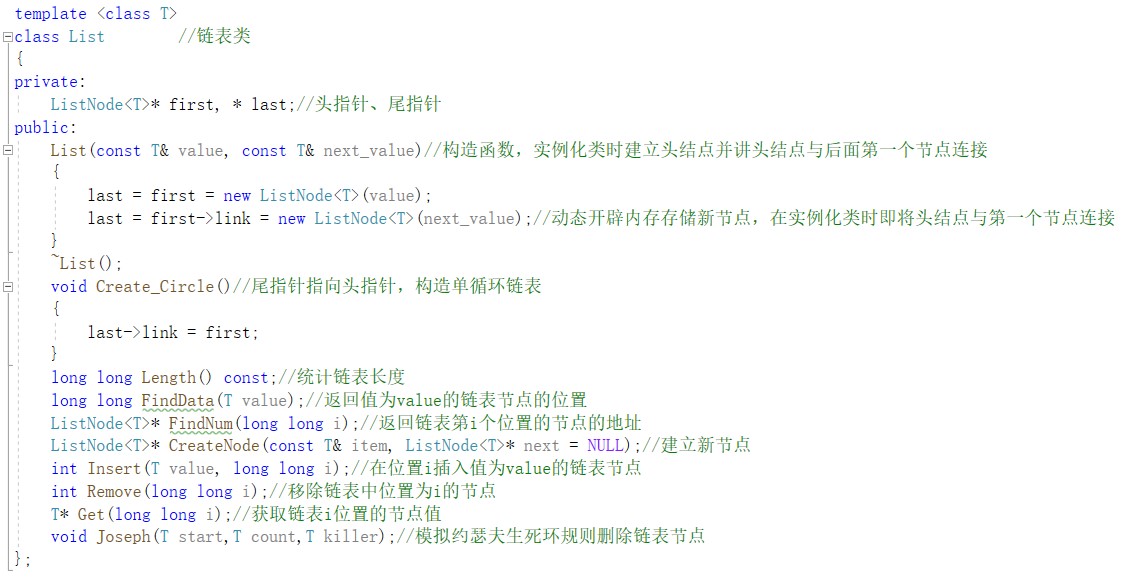
## 屏幕截图 2021-12-19 1316072.2 类结构设计

①链表节点ListNode类：

单循环链表中每个节点需要包含的信息是旅客的序号以及当前节点与后继节点的位置关系。

ListNode节点类被单循环链表类List所运用，进而构成完整的单循环链表。

## ②单循环链表类：



## List带头节点的单循环链表类包含头指针first指向链表头结点，尾指针last则指向链表中的最后一个节点，且尾指针的后继指向头结点，从而形成闭环结构，主要通过实例化类并不断插入新节点来构建链表，通过删除相应位置的节点模拟淘汰某标号的旅客。此外，上述插入与删除功能的实现依赖于寻找某位置的节点、寻找某值的节点位置等功能，下文将对些类函数的**核心代码**进行详细说明：

1. 构造函数：动态开辟内存存储新节点，在实例化类时将头结点与第一个节点连接。
2. 析构函数：功能是删除单循环链表中的所有节点，主体采用循环结构实现。在本程序的单循环链表中，last指向的就是first，因此只要链表中头指针不指向尾指针，即头指针自身，那么链表就不空，就一直循环执行删除操作。具体的删除流程是：用指针q指向头结点的下一个节点，并用q暂存其地址，接着令头结点指向下下个节点，与紧挨头结点的下一个节点脱钩，以便于对其执行删除操作，最后删除q指向的节点。当循环执行完毕后，链表中只余下头结点，将last指针与first指针对齐，最后删除头结点即可完成单循环链表的删除操作。
3. Length函数：功能是返回链表长度，即链表除头结点以外的节点个数，主体采用循环结构实现：首先从头结点指向的下一个节点开始计数，只要节点不空且表示当前位置的p指针没有越过last循环回到first，即未到链表末尾，则一直循环计数，并将指针指向当前节点的下一个节点。最终循环完毕后返回计数结果即可。
4. FindData 函数：

功能：返回值为value的链表节点的位置

主体采用循环结构实现，从头结点指向的下一个节点开始，在链表长度范围内查找，如果找到即可以跳出循环，返回计数值。此外，如果p==NULL或者p越过last循环回到first, 那说明在链表中没找到拥有该值的节点，返回-1，表示未找到。

5.FindNum 函数：

功能：返回链表第i个位置的节点的地址

主体采用循环结构实现，首先对非法的无效参数进行处理：如果传递的参数小于-1，则为非法参数，直接返回NULL，若参数是-1，则返回头结点，否则通过循环，在链表长度范围内一直循环第i个位置或最后一个节点，通过暂存指针p不断指向下一个节点，这样一来，如果第i个位置在链表长度范围内，返回的是节点地址，如i已经超出了链表长度，此时p的值恰好是NULL，返回NULL表示该位置不存在有效节点。

6.CreateNode函数：功能：建立新节点

通过动态内存开辟，new新节点并初始化，参数中应当包含用于初始化新节点的item值以及所指向的下一个节点的地址，这样就可以建立新节点并将新节点指针指向下一个节点。

7. Insert函数：功能：在位置i插入值为value的链表节点

首先在第i个位置插入节点当然需要定位i-1位置的节点，以使其指针指向插入的新节点，接着判断第i-1个节点是否为NULL，是，则表明插入的位置为非法位置，返回0表示插入失败；否则表明插入位置有效，执行后续插入操作：先动态开辟内存存储新建立的节点，并让新建立的节点指向原来在i位置的节点（调用CreateNode函数），接着处理尾结点，如果p已经是尾结点的话，那么新插入的节点成为新的尾结点。最后，原来i-1位置的节点指针指向新建节点，此时，新建节点成为第i位置的节点，原来i位置及后面的节点位置随着这一步整体自然后移1位，最终返回1表示插入成功。

需要注意的是，插入的流程不可随便更改，例如：如果在将原来i-1位置的节点指针指向新建节点之后才处理尾结点，就不能直接通过p->link是否为NULL判断新插入的节点是否应该成为新的尾结点。

8.Remove函数：功能：删除链表中位置为i的节点

首先调用Find(i-1)定位i-1位置的节点，以便于更改其指针指向，接着如果第i-1位置为NULL或者第i位置为NULL,则表明第i位置本身不存在节点，自然就不需要额外的删除操作，返回0,；否则令i-1位置的指针指向下下个节点，跳过要删除的i位置节点，然后判断尾结点情况，如果要删除的节点是尾节点的话，尾结点指向应前移一位，最终调用delete关键字释放建立节点时为其开辟的内存空间，然后返回1表示删除操作执行成功。同样，删除操作各步流程不可随便更改。

9.Get函数：

功能：获取链表i位置的节点值

首先调用Find(i)函数定位到i位置，如果该位置为空，则表明为非法位置，直接返回NULL,否则返回i位置节点的值,即为旅客的编号。

1. Create\_Circle函数：

功能：形成单循环链表结构

此函数仅仅只有一句话：last->link = first;

在建立完所有节点之后，执行此函数，将尾指针指向头指针，即可构成单循环链表。

11.Joseph函数：

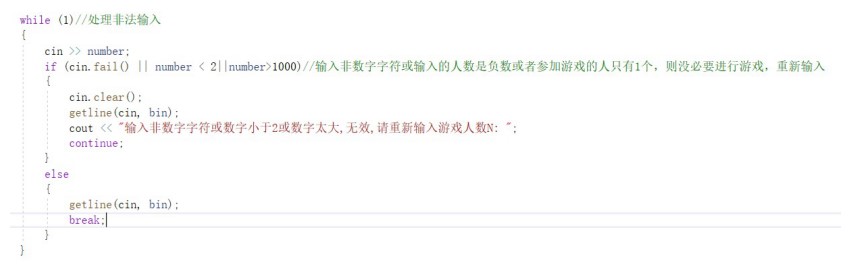
功能：模拟约瑟夫生死环规则删除链表节点并进行相关信息的输出

首先根据用户指定的报数起始位置定位到相应的节点，接着进入双循环结构，外层循环判断通过判断是否到达淘汰人数判断是否继续循环，内层循环通过判断是否达到死亡数字来判断是否对当前节点执行删除操作，若进行删除操作，应该先输出该节点的信息，再进行删除。当双层循环结束之后，调用length函数即可得到最终的幸存者人数。最后遍历单循环链表，逐个输出节点信息，即得到最终的幸存者编号。

# 3. 程序实现流程

首先说明的一点是：综合考虑程序执行时间以及需要的内存开销，本程序对K、M、N、S均限定了范围，防止程序在某些边界条件下宕掉。其中，2<=N<=1000, 1<=S<=N;

2<=M<=2147483647 (int型数据的正数上限); 1<=K<N; 为此，本程序在输入K、M、N、S值时均采用了相似的结构来处理非法的非数字字符输入以及超范围数字输入，接下来以输入N时的代码结构为例进行说明：



循环结构的作用是：如果输入非法的非数字字符或者输入的数字超过了规定的范围，那么就不满足循环的跳出条件，一直执行循环操作，即一直重新输入，直到用户输入了满足条件的数字。

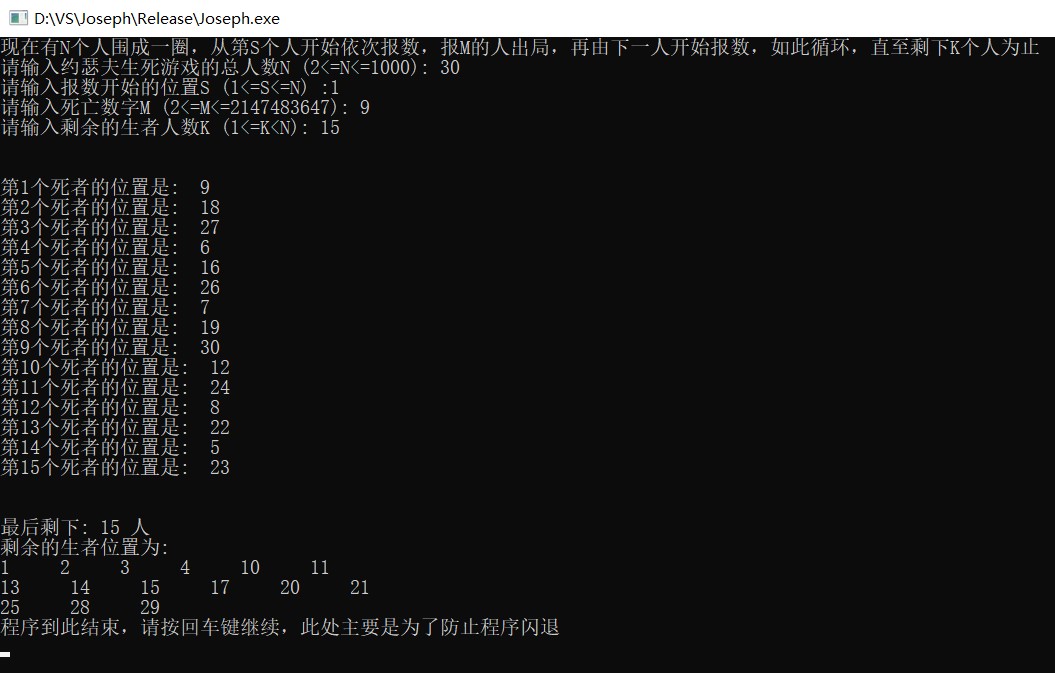
本程序的流程是较为简单的直线型顺序操作，依次由用户输入游戏总人数N、报数起始位置S、死亡数字M、最后的幸存者人数K，接着调用单循环链表类的Insert函数建立相应长度的单循环链表，最终调用Joseph函数模拟约瑟夫生死环游戏的流程输出淘汰者序号与幸存者序号，则程序整体流程结束。

接下来是对程序的测试部分：

# 4.测试

## 4.1 功能测试

# ①30名游戏参与者，从第一个人开始报数，报数9的人出列，最终留下15人



②错误输入处理以及输入超范围数字时测试：

从上到下的错误依次是：

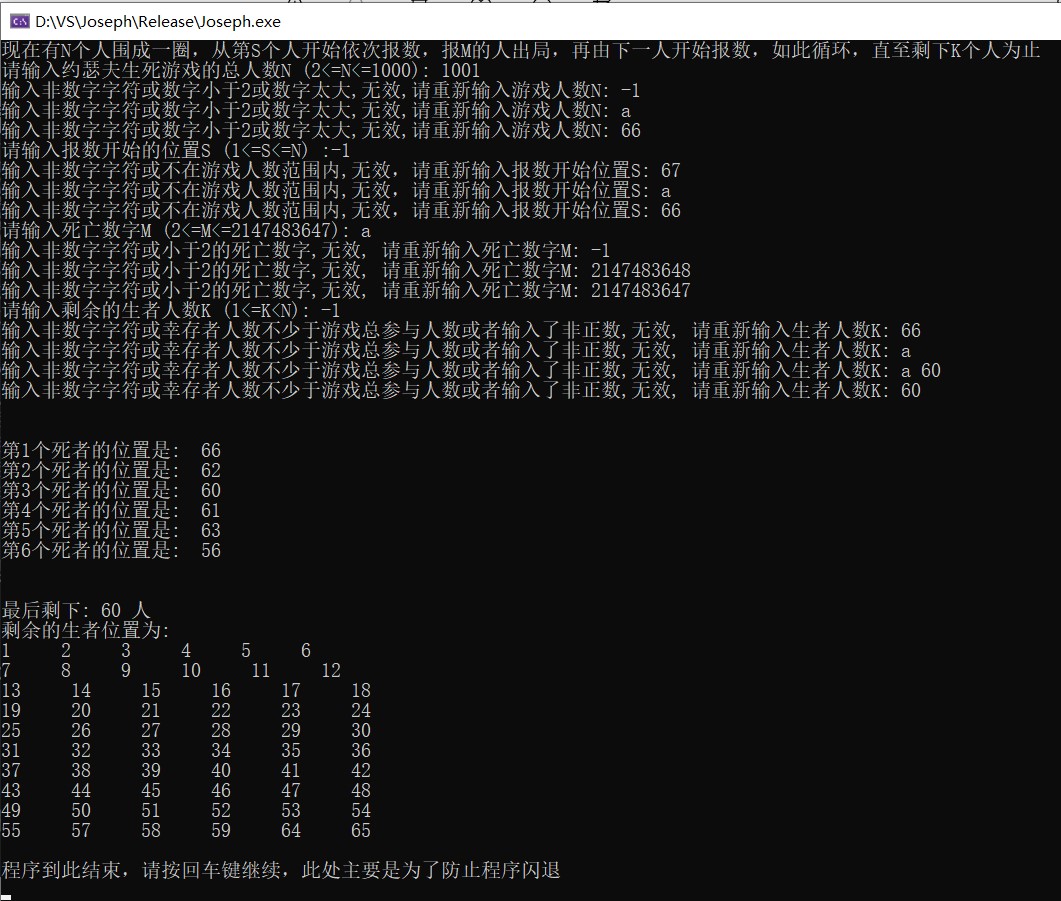
游戏人数N : 游戏人数超范围、输入的人数为负数、输入的人数是非数字字符

报数开始位置S: 输入负数、报数开始位置超出了游戏总人数、输入非数字字符

死亡数字M: 输入非数字字符、输入非正数、超过限定的数字上限

生还者人数K: 输入非正数、输入的人数不少于游戏参与者的总人数、输入非数字字符、输入非数字字符且后面额外尾随了多余的输入

最终输出的是：66名游戏参与者，从第66个位置开始报数，报到2147483647的人出列，最终幸存60人的结果：



③范围边界数字测试：

1000个人参与游戏，从第一个位置开始报数，报到2147483647的人出列，最终剩余999名幸存者（输出内容太多，仅截取了）。

