**数据结构课程设计《约瑟夫生死者游戏》项目说明文档**

2253893 苗君文 软件工程

**1. 项目简介**

**1.1. 项目背景**

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。

**1.2. 项目要求**

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。

本游戏要求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。

本游戏要求输出的内容是包括：

1. 离开旅客的序号；

2. 剩余旅客的序号。

**1.3. 输入格式**

本项目首先需要输入要执行的操作，选择“A”，即可看到游戏简介；选择“Y”，即可进入游戏；选择“N”，即可退出游戏。若进入游戏，则需要输入生死游戏的总人数N，游戏开始的位置S，死亡数字M，剩余的生者人数K。

**1.4. 输出格式**

本项目针对输入输入有详细的输入错误处理，若用户输入错误，则程序会输出相关提示信息，并要求重新输入。而针对用户选择的操作，可以输出相应的操作结果。例如，进入游戏开始用户输入完所需的参数后，将输出所有死者的位置，剩余的生者人数以及剩余的生者的所在位置。

**2. 设计思路**

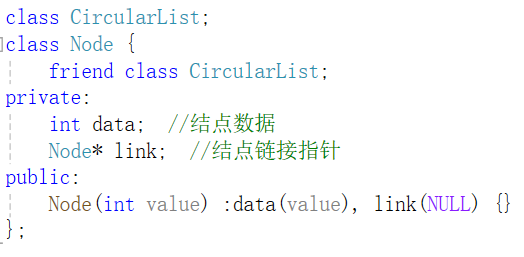
**2.1. 数据结构设计**

在本项目中，采用循环链表以存储相关内容并做出相应的操作。循环链表在约瑟夫问题中有着很好的适用性，特别是在需要反复遍历的情况下，使用这种数据结构的具体原因如下：约瑟夫问题中的出列操作需要循环地遍历链表。使用循环链表可以使得链表的末尾结点指向链表的头结点，形成一个闭环，从而在遍历到链表末尾时能够轻松地回到链表的起始位置，实现循环操作，从而解决约瑟夫问题的实际需求；同时，在约瑟夫问题中，每隔一定数量的结点进行删除。在循环链表中，删除节点的操作更加高效，因为只需要更新前一个结点的指针即可，不需要遍历整个链表，这对于频繁使用删除操作的约瑟夫游戏很有帮助；而且，在游戏的开始阶段，需要将N个旅客按照一定顺序插入到链表中，循环链表在插入结点是更加方便，只需要修改相邻结点的指针即可；最后，使用循环链表非常符合约瑟夫问题的数学特性。

**2.2. 类设计**

为了实现约瑟夫生死者游戏的功能，选择使用链表结点Node类和循环链表CircularList类。Node类负责存储和表示各个旅客的位置信息及其指向的下一个结点位置，而CircularList类提供了对旅客们的各种插入、删除、执行约瑟夫游戏等操作。这两个类之间采用复合方式使用，这种设计使得程序更加模块化，易于维护和扩展。

**2.2.1. Node类（结点类）**



成员变量：

* int data: 存储结点的编号，即旅客的序号。
* Node\* link: 指向下一个结点，形成链表。

成员函数：

* 构造函数：接收一个整数参数（旅客的序号），初始化成员变量，并将 link 设置为 NULL。这样设计使得创建结点时可以直接传入旅客的序号。

**2.2.2. CircularList类（循环链表类）**

成员变量：

* Node\* first: 指向链表的头结点。
* Node\* last: 指向链表的尾结点。

成员函数：

* 构造函数：创建一个空的链表，将 first 和 last 都初始化为 NULL。
* 析构函数：释放链表中所有结点的内存，避免内存泄漏。
* insert插入操作：在链表的最后一个位置插入新的结点。
* display显示链表：输出链表中所有结点的信息。
* playJosephusGame游戏操作：实现约瑟夫生者死者游戏的逻辑。在游戏中，根据输入的参数执行游戏，输出死者的位置，剩余的生者人数，以及剩余的生者位置。
* 获取头结点getFirst，用于方便其他操作进行。
* 获取尾结点getLast，用于方便其他操作进行。

**2.3. 主程序设计**

首先，输出欢迎信息，并让用户选择要执行的操作。根据不同的选项执行相应的内容，若选择“游戏简介”则输出简介；若选择“退出游戏”，则提示相关信息并退出；若选择“开始游戏”，则先创建一个空的CircularList类型的链表list，并用循环以形成约瑟夫环，再调用playJosephusGame函数完成约瑟夫生死者游戏。对应的操作结束后，用户可以继续选择相应的游戏操作，循环往复。

**2.4.** **输入错误处理设计**

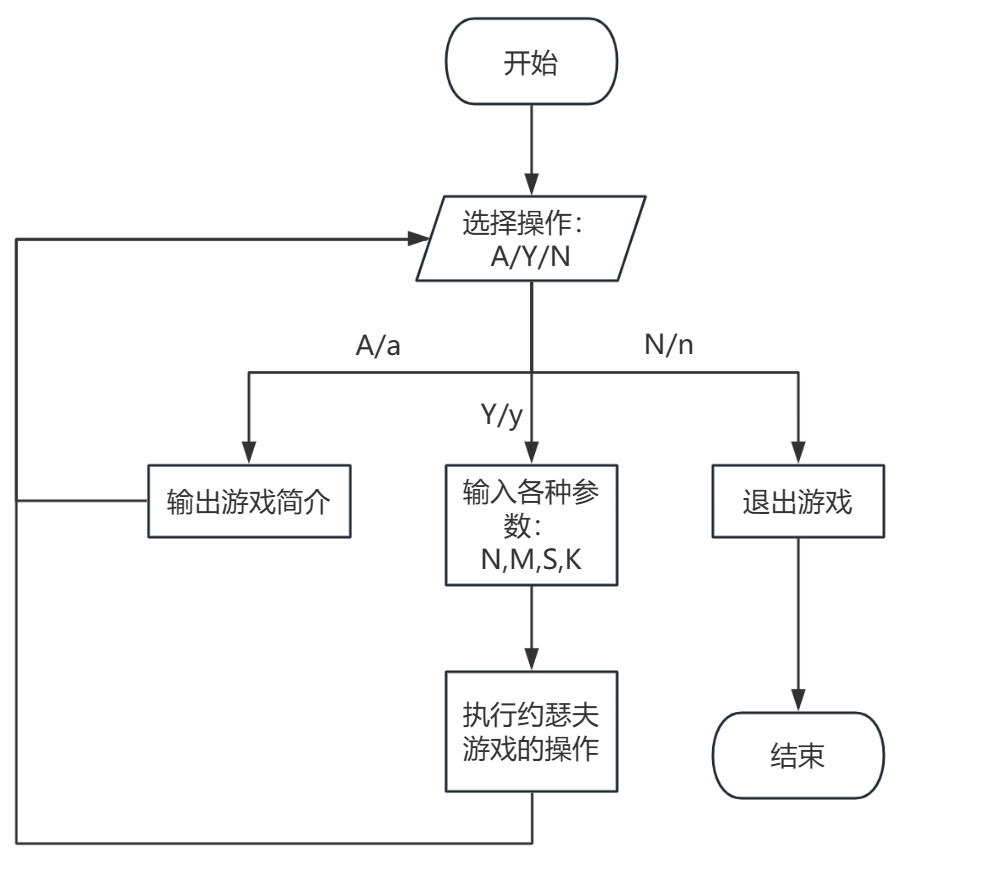
**2.4.1. 单个参数的输入错误处理**

函数dealInputError使用 while 循环不断尝试获取用户输入，直到输入满足要求。使用 cin.fail() 及min、max来判断输入是否出错，检查输入是否在有效范围内。如果输入无效，则输出错误信息，清除输入缓冲区并忽略之后的字符，并重新输出输入的提示信息。通过一个字符来获取输入一个数之后的字符以检查输入字符的个数，若不正确，也输出错误信息，并重新输出输入的提示信息，执行相应清除操作。如果输入有效，则跳出循环。

需要注意的是，在输入游戏开始的位置S、死亡数字M、剩余的生者人数K时，输入的数值大小不得超过生死游戏的总人数N，否则需要重新输入。

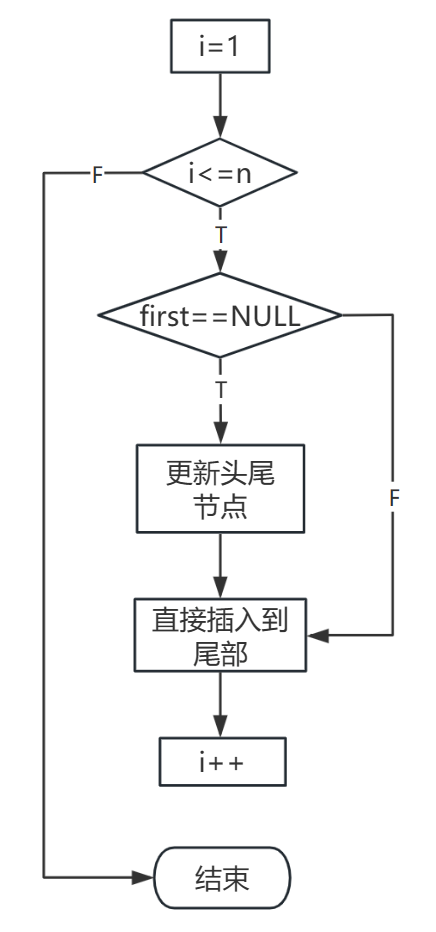
**3. 功能实现**

**3.1. 主体功能实现**



**3.2. 操作选项Y的实现**

**3.2.1. 使用insert函数插入旅客**



**3.2.2. 约瑟夫游戏功能的实现**

该功能实现通过CircularList类中的playJosephusGame函数实现。该函数使用两个指针 current 和 prev 分别指向当前结点和当前结点的前一个结点。通过循环将 current 移动到游戏开始的位置（第s个结点）。

接下来，通过两层循环模拟游戏的进行：外层循环（i从0到n-k-1）：表示淘汰（死亡）的次数，总共进行n-k轮淘汰。内层循环（j从0到m-1）：表示数到第m个结点的过程。

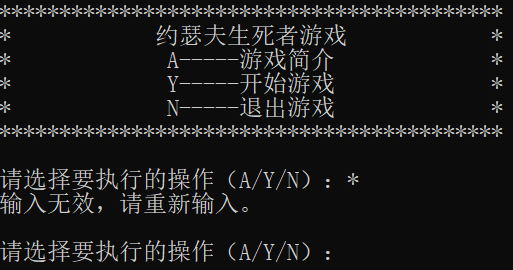
在内层循环中，通过移动 current 和 prev 实现找死者的过程，找到第m个结点。输出第i+1个被淘汰者的位置（current->data）。

根据被淘汰者的位置，从链表中删除该结点：若被淘汰者是第一个结点（current == first），更新链表的头结点（first = current->link）。若被淘汰者是最后一个结点（current == last），更新链表的尾结点（last = prev）。删除结点后，释放被淘汰者结点的内存。最后，输出最终剩余的生者人数（k），并调用链表的 display 方法展示剩余生者的位置。

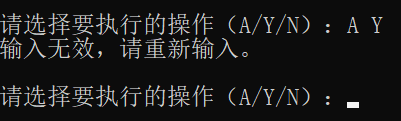
**4. 测试结果**

**4.1. 选择要执行的操作**

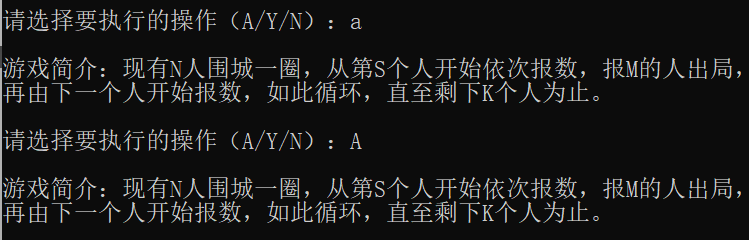
（1）输入的内容非法



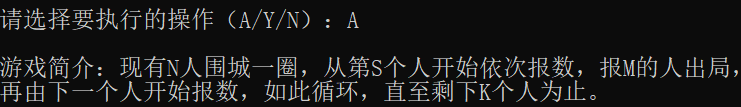
（2）输入的个数超过一个



（3）输入正确即可进入相应的操作（大小写均可）



**4.2. 游戏简介（选项A）**



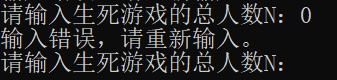
**4.3. 开始游戏（选项Y）**

**4.3.1. 输入生死游戏的总人数N**

（1）输入的类型非法



（2）输入的范围非法



（3）输入正确即可进入下一环节

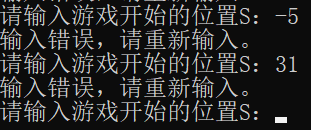


**4.3.2. 输入游戏开始的位置S**

（1）输入的类型非法



（2）输入的范围非法，此处S不可超过N



（3）输入正确即可进入下一环节

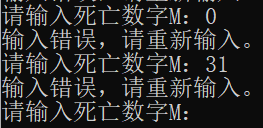


**4.3.3. 输入死亡数字M**

（1）输入的类型非法



（2）输入的范围非法，此处M不可超过N



（3）输入正确即可进入下一环节

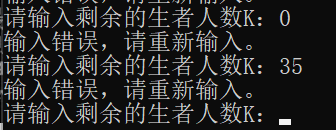


**4.3.4. 输入剩余的生者人数K**

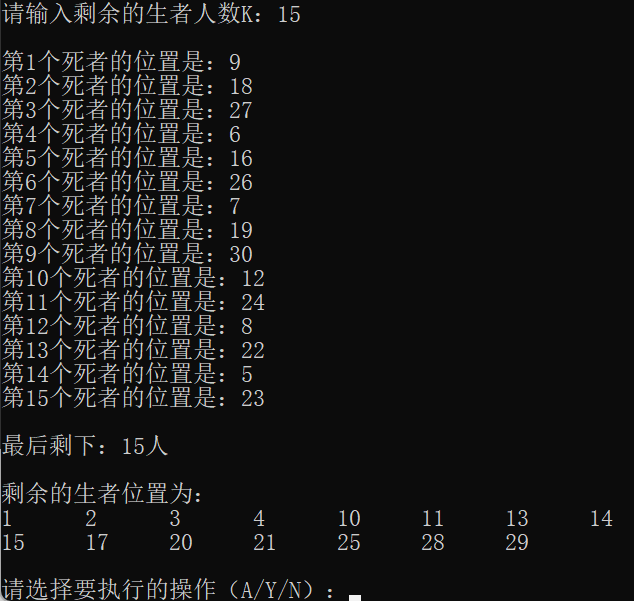
（1）输入的类型非法



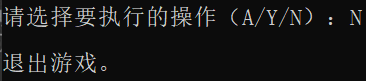
（2）输入的范围非法，此处M不可超过N



（3）输入正确即可输出死者和最后的生者



**4.4. 退出游戏（操作N）**



**4.5. 程序的测试结果总览（仅展示输入正确有效的情况）**

