数据结构课程设计

项目说明文档

约瑟夫生死游戏

|  |  |
| --- | --- |
| 作者姓名： | 杨烜赫 |
| 学 号： | 2252709 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |



同济大学

Tongji University

1. **题目简介**

**1.实验背景**

约瑟夫问题（Josephus Problem）源自一个著名的历史故事，通常与古代犹太历史学家弗拉维奥·约瑟夫斯（Flavius Josephus）的一个经历有关。这个问题在数学和计算机科学领域中作为一个著名的逻辑和算法问题被广泛研究。据历史记载，约瑟夫斯是一位犹太历史学家，也是犹太-罗马战争时期的一个军事指挥官。在一次战争中，约瑟夫斯和他的40多个同伴被罗马军队包围在洞穴中。面对绝望的形势，他们决定自杀而不是投降。然而，自杀的方式引发了争议。为了公平地决定谁将会被杀，他们采用了一种特殊的方法：所有人围成一个圈，然后从某个特定的起点开始数数。每数到第N个人，那个人就必须自杀。这个过程一直持续，直到只剩下最后一个人。约瑟夫斯并不想死，他迅速计算出了一个安全位置，并最终存活下来，成为了这个故事的传说。后来，这个问题在数学上被抽象为约瑟夫问题或约瑟夫环，成为了组合数学和算法设计中的一个经典问题。在数学和计算机科学中，约瑟夫问题经常被用作算法设计的练习题，特别是与递归算法、循环链表和队列等数据结构的应用相关。这个问题不仅是一个有趣的数学难题，而且在实际编程中也有一定的应用价值。

**2.实验功能**

项目功能要求：(要求采用单循环链表)

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。

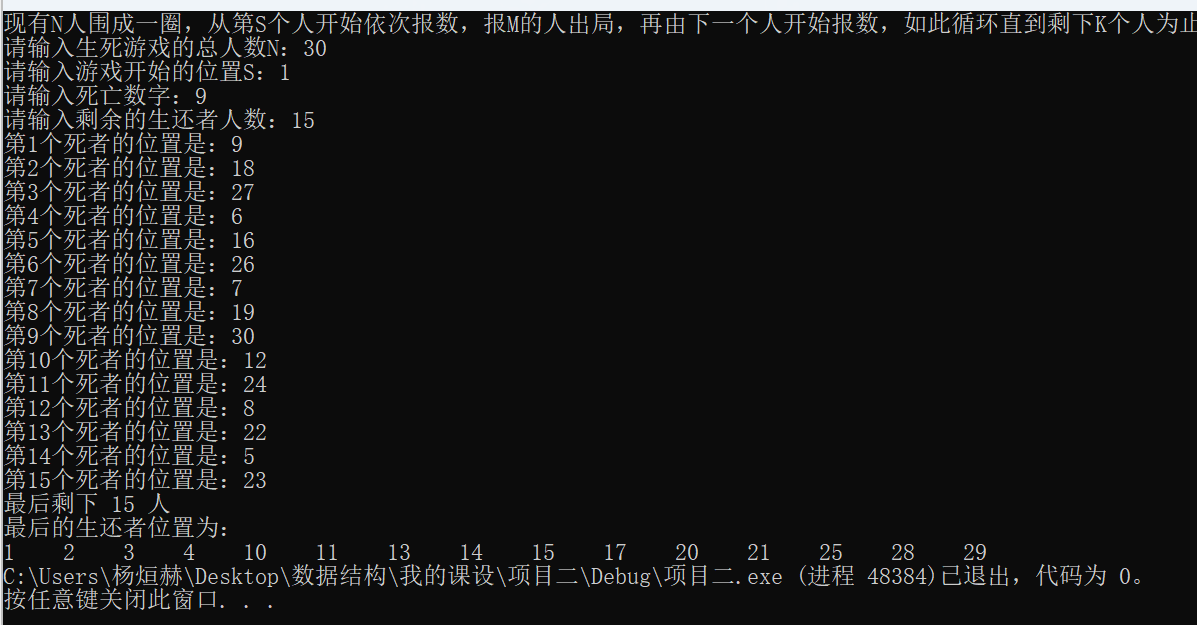
本游戏要求用户输入的内容包括：

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。

**3.项目示例**



1. **项目设计**

## 1.数据结构设计

本项目采用循环链表数据结构来实现约瑟夫问题。循环链表是一个单向链表，其中每个节点指向下一个节点，并且最后一个节点再次指向第一个节点，形成一个闭环。这种数据结构非常适合约瑟夫问题，因为它允许连续遍历所有节点，而无需重新开始。

节点定义：链表的每个节点由结构circ\_linknode表示，包含一个表示位置的pos整数和指向下一个节点的next指针。

## 2.类结构设计

类定义：circ\_list

目的：管理循环链表，并实现约瑟夫问题的解决方案。

成员变量

circ\_linknode\* first：

类型：指向 circ\_linknode 类型的指针。

用途：指向循环链表的第一个节点，作为链表的头节点。

构造函数

circ\_list()：

功能：构造函数，用于初始化 circ\_list 实例。

实现：将 first 成员变量初始化为 nullptr，表示链表开始时为空。

成员函数

初始化链表 - bool Init\_circ\_list(int n)：

参数：int n - 链表中的节点数量。

返回值：bool - 表示链表是否成功初始化。

功能：创建一个包含 n 个节点的循环链表。每个节点的 pos 成员按顺序赋值，最后一个节点的 next 指向第一个节点，形成循环。

异常处理：如果内存分配失败，输出错误信息并返回 false。

执行约瑟夫问题算法 - bool circ\_delete(int n, int s, int m, int k)：

参数：

int n - 链表中的节点总数。

int s - 开始报数的位置。

int m - 每次报数的间隔。

int k - 最终留在链表中的节点数量。

返回值：bool - 表示操作是否成功执行。

功能：从第 s 个节点开始，每数到第 m 个节点就从链表中删除该节点，直到链表中剩下 k 个节点为止。期间打印出被删除的节点位置和最后剩余的节点位置。

异常处理：如果参数无效或链表未初始化，输出错误信息并返回 false。

析构函数

~circ\_list()：

功能：析构函数，用于在对象销毁时清理资源，例如释放链表中所有节点所占用的内存。通过这种类结构设计，可以有效地建立和管理一个高效、灵活的考生信息链表，满足项目的各项功能需求。

## 3.成员与操作设计

在约瑟夫问题的程序实现中，circ\_list 类的成员与操作设计如下：

**成员变量**

first：类型为 circ\_linknode\*，是指向循环链表的第一个节点的指针。它作为链表的头节点，是整个链表结构的起点。

**构造函数**

circ\_list()：初始化 first 为 nullptr，表示链表在初始状态下为空。

**成员函数**

Init\_circ\_list(int n)：

目的：初始化一个包含 n 个节点的循环链表。

参数：n - 链表中节点的数量。

返回值：布尔值，表示链表是否成功初始化。

功能：创建 n 个新的 circ\_linknode 节点，并将它们连接成一个循环链表。每个节点的 pos 按顺序赋值，从1开始。

circ\_delete(int n, int s, int m, int k)：

目的：根据约瑟夫问题的规则移除节点。

**参数：**

n - 初始时链表中的节点总数。

s - 开始报数的起始位置。

m - 报数的间隔，每数到 m 就移除一个节点。

k - 最终在链表中保留的节点数量。

返回值：布尔值，表示操作是否成功执行。

功能：从第 s 个节点开始报数，每数到第 m 个节点时删除该节点，直到链表中剩下 k 个节点。期间，打印出每个被删除的节点的位置及最后剩余节点的位置。

异常处理：如果输入参数无效或链表未正确初始化，则输出错误信息并返回 false。

**析构函数**

~circ\_list()：

目的：释放循环链表占用的所有资源。

1. **项目实现**

**简介**

本项目实现了约瑟夫问题的解决方案，使用循环链表模拟了一个固定数量的人围成一圈的场景，其中通过特定的规则来逐个移除人员，直至剩下指定数量的人。这个问题的实现主要包括初始化循环链表、执行约瑟夫问题的算法，并在过程中提供反馈。

**3.1. 初始化循环链表**

功能：建立一个由n个节点组成的循环链表。

1. **bool** circ\_list::Init\_circ\_list(**int** n) {
2. **if** (n <= 0) {
3. cerr << "链表节点数量必须大于0" << endl;
4. **return** **false**;
5. }
6. first = **new** circ\_linknode;
7. first->pos = 1;
8. first->next = first;
9. circ\_linknode\* pcurrent = first;
10. **for** (**int** i = 2; i <= n; i++) {
11. circ\_linknode\* pnew = **new** circ\_linknode;
12. pnew->pos = i;
13. pnew->next = first;
14. pcurrent->next = pnew;
15. pcurrent = pnew;
16. }
17. **return** **true**;
18. }

Init\_circ\_list 方法用于初始化约瑟夫问题的核心数据结构——一个循环链表。该方法创建一个由用户指定数量的节点组成的闭环链表，其中每个节点代表一个位置（或一个参与者）。方法的关键在于正确地建立链表的循环结构，即确保链表的最后一个节点指回第一个节点，形成一个环。

**健全性测试：**

**输入验证：**

方法首先要求用户输入链表的节点数量。如果输入的数量小于或等于0，这显然是无效的，因为链表至少需要一个节点才有意义。

**内存分配检查：**

对于每个新创建的节点，方法会检查是否成功分配了内存。如果 new 操作失败（例如由于内存不足），程序将输出错误信息，并退出方法。这可以防止程序在无效的内存位置上执行操作，避免了潜在的崩溃。

**循环链表构建：**

方法正确地将每个新节点加入到链表中，并确保链表的最后一个节点指向第一个节点。这样的循环结构对于实现约瑟夫问题的逻辑至关重要。

通过 Init\_circ\_list 方法，程序能够创建并初始化一个循环链表，为实现约瑟夫问题提供了必要的基础。同时，通过对输入进行验证和对内存分配进行检查，该方法确保了程序的健壮性和稳定性。这些错误处理机制不仅提高了程序的可靠性，也提升了用户体验，因为它们防止了程序因无效输入或内存问题而异常终止。、

**3.2. 执行约瑟夫问题算法**

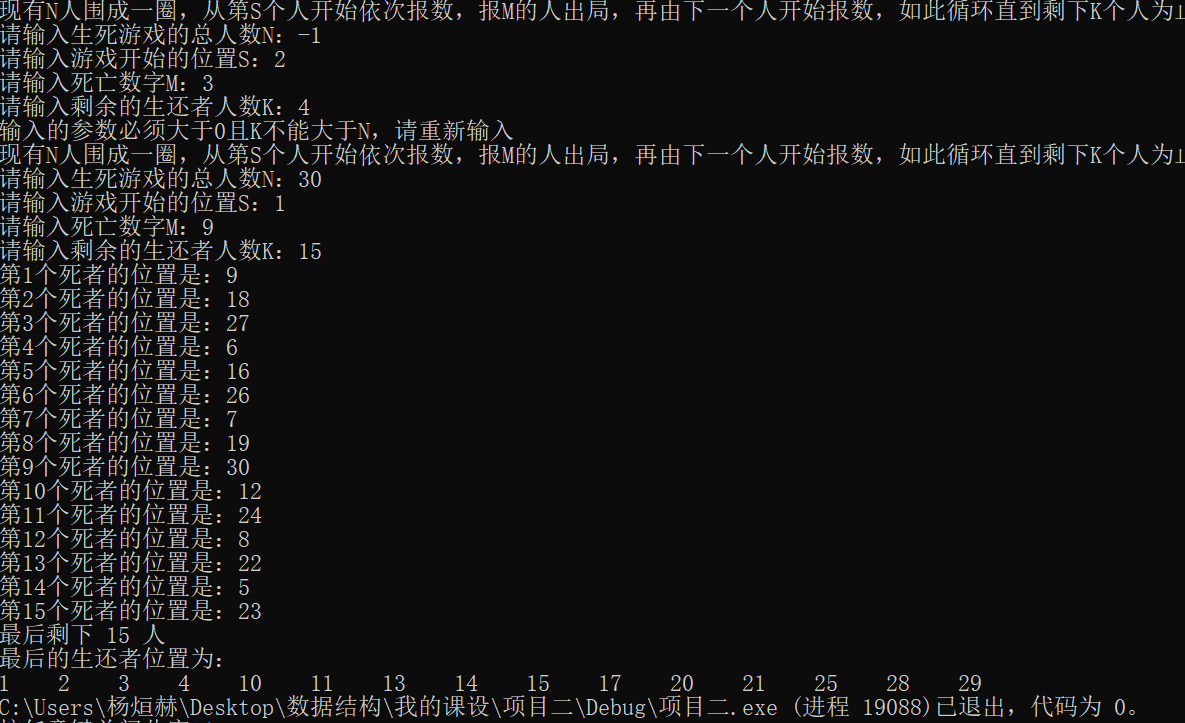
1. **bool** circ\_list::circ\_delete(**int** n, **int** s, **int** m, **int** k)
2. {
3. **if** (n <= 0 || s <= 0 || m <= 0 || k <= 0) {
4. cerr << "参数必须大于0" << endl;
5. **return** **false**;
6. }
7. **if** (first == nullptr) {
8. cerr << "链表未初始化" << endl;
9. **return** **false**;
10. }
11. circ\_linknode\* current = first;
12. **int** num = n;
13. **int** total = 0;
14. **for** (**int** i = 0; i < s - 1; i++)
15. {
16. current = current->next;
17. }
18. **while** (num > k)
19. {
20. **for** (**int** i = 1; i < m - 1; i++)
21. {
22. current = current->next;
23. }
24. circ\_linknode\* temp = current->next;
25. current->next = temp->next;
26. total++;
27. cout << "第" << total << "个死者的位置是：" << temp->pos << endl;
28. **delete** temp; // 释放节点内存
29. num--;
30. current = current->next;
31. }
32. cout << "最后剩下 " << k << " 人" << endl;
33. cout << "最后的生还者位置为：" << endl;
34. current = first;
35. **for** (**int** i = 0; i < k; i++) {
36. cout << current->pos << "    ";
37. current = current->next;
38. }
40. // 释放头节点内存
41. **delete** first;
42. first = nullptr;
43. **return** **true**;
44. }
46. circ\_list Josephus;
47. **int** main() {
48. **int** N, S, M, K;
50. **while** (**true**) {
51. cout << "现有N人围成一圈，从第S个人开始依次报数，报M的人出局，再由下一个人开始报数，如此循环直到剩下K个人为止" << endl;
52. cout << "请输入生死游戏的总人数N：";
53. cin >> N;
54. cout << "请输入游戏开始的位置S：";
55. cin >> S;
56. cout << "请输入死亡数字M：";
57. cin >> M;
58. cout << "请输入剩余的生还者人数K：";
59. cin >> K;
60. **if** (N > 0 && S > 0 && M > 0 && K > 0 && K <= N) {
61. **break**; // 当所有输入都合理时，跳出循环
62. }
63. **else** {
64. cerr << "输入的参数必须大于0且K不能大于N，请重新输入" << endl;
65. cin.clear(); // 清除输入缓冲区的错误标记
66. cin.ignore(1024, '\n'); // 忽略缓冲区剩余的输入
67. }
68. }
70. circ\_list Josephus;
71. Josephus.Init\_circ\_list(N);
72. Josephus.circ\_delete(N, S, M, K);
74. **return** 0;
75. }

**主要包含两个核心部分：**

初始化循环链表（Init\_circ\_list方法）：创建并初始化一个包含N个节点的循环链表，每个节点代表链表中一个人的位置。

执行约瑟夫问题的核心算法（circ\_delete方法）：从指定的起始位置S开始，按照给定的间隔M移除节点，直到链表中剩下K个节点。

用户通过命令行输入参数N（总人数）、S（起始位置）、M（报数间隔）、K（剩余人数），然后程序根据这些参数执行约瑟夫问题的逻辑。

**健全性测试：**

**输入验证：**

在主函数中，通过一个循环确保用户输入的N、S、M、K都是合理的值。不合理的输入会导致程序提示错误并要求重新输入。

**链表初始化检查：**

circ\_delete 方法在执行之前检查链表是否已初始化。如果未初始化（first为nullptr），程序将输出错误信息并返回false。

**参数合法性检查：**

circ\_delete 方法还检查输入参数的合法性。任何参数如果小于或等于0，或者K大于N，方法会输出错误信息并返回false。

**内存管理：**

在创建循环链表时，为每个新节点动态分配内存。在链表不再需要时，释放这些内存以避免内存泄漏。

**循环逻辑处理：**

circ\_delete 方法中的循环逻辑正确地处理了从指定位置开始的计数和节点的移除，确保了算法的正确实现。

**异常输入处理：**

在主函数中，如果用户输入的任何一个参数不符合要求，程序将清除输入缓冲区的错误状态并忽略掉缓冲区中剩余的输入，然后提示用户重新输入，从而避免了程序因错误输入而崩溃。

**四、程序创新点**

**4.1循环链表的高效应用：**

本实验创新地采用循环链表来模拟约瑟夫问题，相比于传统的数组或列表方法，循环链表在处理大量数据和频繁删除操作时更为高效。

**4.2动态输入验证与异常处理：**实验中引入了对用户输入的动态验证机制，确保输入的有效性和合理性。当输入不符合要求时，系统会提示错误并要求重新输入，大大增强了程序的健壮性和用户体验。

**4.3清晰的用户交互设计**：程序通过命令行界面提供清晰的用户指引和提示，使得用户操作直观易懂，从而提高了程序的可访问性和易用性。

**4.4优化的内存管理**：在链表节点创建和删除过程中，实验对内存管理进行了细致的处理，确保了程序运行时的稳定性和效率。

**五、实验小结**

本实验成功地实现了约瑟夫问题的计算机模拟，使用循环链表作为核心数据结构。这种方法不仅提高了处理效率，也使得问题的解决更加直观和易于理解。实验过程中，特别注意到了用户输入的验证和异常处理，确保了程序能够稳定运行，即使在面对非预期输入时也能妥善处理。通过这个项目，我们深入理解了循环链表的实际应用，尤其是在处理循环数据结构和频繁删除操作时的优势。此外，该实验也锻炼了我们在程序设计、错误处理和用户交互设计方面的能力。这些技能对于软件开发和算法设计具有重要意义。