华东师范大学计算机科学与技术学院上机实践报告

课程名称：数据结构	**年级**：2022 级	**上机实践成绩**：
指导教师：金健	**姓名**：岳锦鹏	**上机实践时间**：2 学时
上机实践名称：第二章作业	**学号**：10213903403	**上机实践日期**：2023/10/10

一、 实验目的

1. 使用第二章知识解决约瑟夫环问题（复杂版）。

二、 实验内容

1. 在原有约瑟夫环的基础上，输入参与这场游戏的人数 n 与自杀报数 m，输出为最后两个幸存者的位置序号。
2. 在第一题的基础上，输入参与这场游戏的人数 n 与两个自杀报数 a 和 b，按照一次 a 一次 b 的顺序自杀（a，b 可以为负数，正数代表正向，负数代表反向），输出为最后两个幸存者的位置序号。

三、 实验原理

1. 程序设计原理。

四、 实验步骤

1. 问题抽象
2. 编写程序
3. 调试程序
4. 完善总结

五、 调试过程、结果和分析

1. 简单考虑，暂时不考虑输入错误或者输入为 0 的情况；
2. JavaScript 没有指针的概念，所以使用对象的引用代替；
3. JavaScript 在严格模式下暂时无法直接删除对象，因此移除结点时无法完全销毁一个节点；
4. 最后只留下两个，而不是 $\min(\text{abs}(a), \text{abs}(b)) - 1$ 个；
5. 删除的结点应该作为第 0 个，此时往前 a 个或往后 b 个，一开始如果 a 是正数那么应该按照 $1, 2, 3, \ldots$ 的顺序遍历，如果 a 是负数那么应该按照 $n, n-1, n-2, \ldots$ 的顺序遍历；
6. 由于此问题中只涉及到初始化链表、删除单个结点两个操作，因此链表的添加结点、清空、销毁等操作暂不需要实现。

六、 总结

1. 由于 JavaScript 的异步性质，需要进行连续多个输入时，可以将回调过程封装成 Promise 对象后，使用 async 创建异步函数，在其中使用 await 可以方便使用 Promise 对象，避免大量回调，同时也可以结合循环、分支等结构，实现更复杂的控制；
2. 抽象出了结点对象和链表对象，并且实现了从当前节点开始寻找第 n 个结点（可以为负数）的方法；
3. 整个文件中只有 input 和 main_ 函数与 Node.js 环境耦合，即使用了 readline 模块，而主要功能放在 main 函数中，因此在浏览器环境中可以直接调用 main 函数，方便代码复用。

七、 附件

1. 在原有约瑟夫环的基础上，输入参与这场游戏的人数 n 与自杀报数 m，输出为最后两个幸存者的位置序号。
   （JavaScript，Node.js 环境）

```
const readline = require("readline");
const rl = readline.createInterface(process.stdin, process.stdout);

class Node {
    previous;
    value;
    next;
}

class LinkList {
    head;
    length;
    constructor(length) {
        // 还是需要头结点，此时 head 表示其中第 0 个结点，length 必须大于 0
        this.length = length;
        this.head = new Node();
        // this.head.value = 1;
        let current, previous = this.head;
        for (let i = 1; i <= this.length; i++) {
            current = new Node();
            current.value = i;
            previous.next = current;
            current.previous = previous;
            previous = current;
        }
        current.next = this.head.next;
        this.head.next.previous = current;
        // this.head.next = this.current; // 不能多重赋值，赋值语句会返回
        ↪ undefined
        this.head.previous = current;
```

```
30      }
31      remove(node) {
32          node.previous.next = node.next;
33          node.next.previous = node.previous;
34          // delete node; // JavaScript 里应该无法直接删除
35          this.length--;
36          return true;
37      }
38
39      // 找到从当前结点开始第 n 个结点（可以为负数），当前结点是第 0 个
40      // 并且再向后返回一个结点
41      get_n_after_node(node, n) {
42          let property_name = "next";
43          if (n < 0) {
44              property_name = "previous";
45              n = -n;
46          }
47          if (this.length < 3) return false;
48          for (let i = 0; i < n; i++) { // 向后走 n 个，例如从 0 开始，到 3 结
            ↪ 束
49              node = node[property_name];
50          }
51          return node;
52      }
53  }
54
55  function main(people_num, num1) {
56      let link_list = new LinkList(people_num);
57      let current = link_list.head;
58      let temp;
59      while (true) {
60          temp = link_list.get_n_after_node(current, num1);
61          if (!temp) break;
62          current = temp;
63          link_list.remove(current);
64      }
65      // 此时 current 是最后一个被删掉的结点，它的 next 和 previous 都是存在的
        ↪ 结点
66      current = current.next;
67      link_list.head.next = current;
68      for (let i = 0; i < link_list.length; i++) {
69          console.log(current.value);
```

```
            current = current.next;
        }
    }

    function input(prompt) {
        return new Promise((resolve) => {
            rl.question(prompt, (answer) => {
                resolve(answer);
            });
        });
    }

    async function main_() {
        people_num = parseInt(await input(" 输入人数"));
        num1 = parseInt(await input(" 输入第一个数字"));
        main(people_num, num1);
        rl.close();
    }

    main_()

    // 输入人数 41
    // 输入第一个数字 3
    // 16
    // 31
```

2. 在第一题的基础上，输入参与这场游戏的人数 n 与两个自杀报数 a 和 b，按照一次 a 一次 b 的顺序自杀（a，b 可以为负数，正数代表正向，负数代表反向），输出为最后两个幸存者的位置序号。

（JavaScript，Node.js 环境）

```
const readline = require("readline");
const rl = readline.createInterface(process.stdin, process.stdout);

class Node {
    previous;
    value;
    next;
}

class LinkList {
    head;
```

```
12      length;
13      constructor(length) {
14          // 还是需要头结点，此时 head 表示其中第 0 个结点，length 必须大于 0
15          this.length = length;
16          this.head = new Node();
17          // this.head.value = 1;
18          let current, previous = this.head;
19          for (let i = 1; i <= this.length; i++) {
20              current = new Node();
21              current.value = i;
22              previous.next = current;
23              current.previous = previous;
24              previous = current;
25          }
26          current.next = this.head.next;
27          this.head.next.previous = current;
28          // this.head.next = this.current; // 不能多重赋值，赋值语句会返回
            ↪ undefined
29          this.head.previous = current;
30      }
31      remove(node) {
32          node.previous.next = node.next;
33          node.next.previous = node.previous;
34          // delete node; // JavaScript 里应该无法直接删除
35          this.length--;
36          return true;
37      }
38
39      // 找到从当前结点开始第 n 个结点（可以为负数），当前结点是第 0 个
40      // 并且再向后返回一个结点
41      get_n_after_node(node, n) {
42          let property_name = "next";
43          if (n < 0) {
44              property_name = "previous";
45              n = -n;
46          }
47          if (this.length < 3) return false;
48          for (let i = 0; i < n; i++) { // 向后走 n 个，例如从 0 开始，到 3 结
            ↪ 束
49              node = node[property_name];
50          }
51          return node;
```

```
52      }
53  }
54
55  function main(people_num, num1, num2) {
56      let link_list = new LinkList(people_num);
57      let current = link_list.head;
58      let temp;
59      while (true) {
60          temp = link_list.get_n_after_node(current, num1);
61          if (!temp) break;
62          current = temp;
63          link_list.remove(current);
64          temp = link_list.get_n_after_node(current, num2);
65          if (!temp) break;
66          current = temp;
67          link_list.remove(current);
68      }
69      // 此时 current 是最后一个被删掉的结点，它的 next 和 previous 都是存在的
        ↪ 结点
70      current = current.next;
71      link_list.head.next = current;
72      for (let i = 0; i < link_list.length; i++) {
73          console.log(current.value);
74          current = current.next;
75      }
76  }
77
78  function input(prompt) {
79      return new Promise((resolve) => {
80          rl.question(prompt, (answer) => {
81              resolve(answer);
82          });
83      });
84  }
85
86  async function main_() {
87      people_num = parseInt(await input(" 输入人数"));
88      num1 = parseInt(await input(" 输入第一个数字"));
89      num2 = parseInt(await input(" 输入第二个数字"));
90      main(people_num, num1, num2);
91      rl.close();
92  }
```

```
93
94  main_()
95
96  // 输入人数 10000
97  // 输入第一个数字 123
98  // 输入第二个数字-654
99  // 1299
100 // 4114
101
102 // 输入人数 99999
103 // 输入第一个数字-848
104 // 输入第二个数字-351
105 // 80178
106 // 23680
```