实验报告（一）：加里森的任务

班级：2022217803
姓名：密言
学号：2022210064

2024.3.20

1 需求分析

1.1 题目

有 n 个加里森敢死队的队员要炸掉敌人的一个军火库，谁都不想去，队长加里森决定用轮回数数的办法来决定哪个战士去执行任务。规则如下：如果前一个战士没完成任务，则要再派一个战士上去。现给每个战士编一个号，大家围坐成一圈，随便从某一个编号为 x 的战士开始计数，当数到 y 时，对应的战士就去执行任务，且此战士不再参加下一轮计数。如果此战士没完成任务，再从下一个战士开始数数，被数到第 y 时，此战士接着去执行任务。以此类推，直到任务完成为止。
加里森本人是不愿意去的，假设加里森为 1 号，请你设计一程序为加里森支招，求出 n,x,y 满足何种条件时, 加里森能留到最后而不用去执行任务。

1.2 分析

分析题目可知，本题要求我们求出在加里森留到最后的情况下，n，x，y 之间的关系。在程序中，我们可以输入一个确定的 n 值，用首尾相接的链表模拟轮回数数的情形，遍历所有的情况（即所有 x,y 的可能取值），输出加里森留到最后时所对应的 x,y 值。
输入 n 的范围为 $n > 0$，正确的输入（例如输入 $n = 5$）会得到以下输出：

```
x and y:
x = 1, y = 4
x = 2, y = 1
x = 3, y = 3
x = 4, y = 2
x = 5, y = 5
```

若输入的 $n <= 0$，会输出”Error data!”，若输入 $n = 1$，则加里森本人就是最后一个人，输出”Is survivor”。

2 概要设计

该问题可以使用首尾相接的循环链表来实现，链表的每一个节点是结构体类型，储存该节点的编号以及下一个节点的地址。首先创建一个编号从 1 到 n、首尾相接的循环链表，并返回头节点地址。然后从头节点地址出发，找到编号为 x 的节点，每次向后遍历 y 个节点，将第 y 个节点从链表中删除，直到链表中只剩下一个节点。返回该节点的编号，如果编号为 1，说明最后剩下的是 1 号队员，符合题目条件，输出对应的 x,y 值。遍历所有可能的 x,y 值，即可找出满足题目条件的所有情况。

程序设计了若干函数，create(n) 用于创建一个编号从 1 到 n 的循环链表，并返回头节点的地址；deleteNode(head,nodeToDelete) 用于删除地址为 nodeToDelete 的节点并返回它的后继节点地址；isSurvivor(n,x,y) 用于判断目前的 n,x,y 值是否能使加里森活到最后，是返回 1，否返回 0。各函数间的调用关系如下图所示。

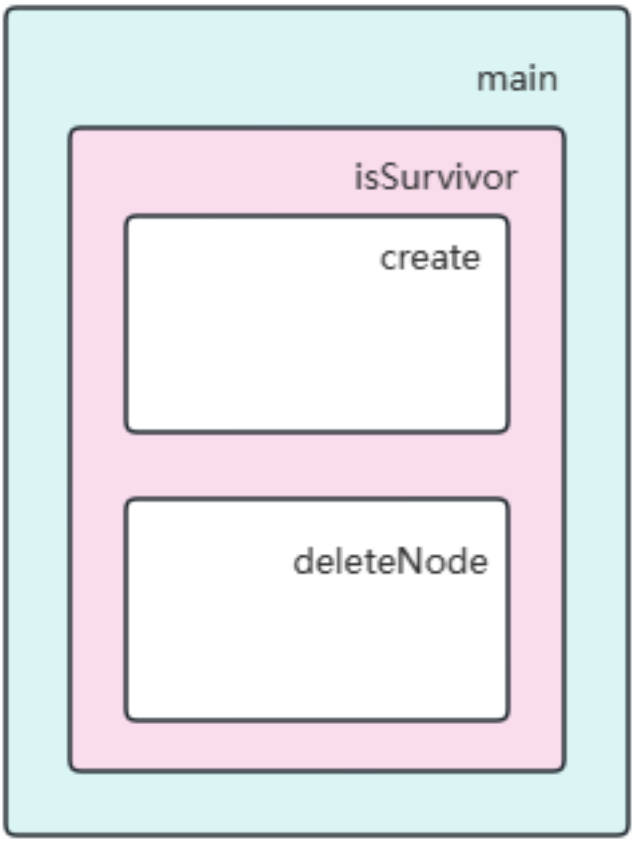

图 1: 函数调用关系

3 详细设计

3.1 结构体节点

结构体中定义一个 int 类型的变量 number，存放节点编号；定义一个结构体指针类型的 next，存储下一个节点的地址。

```
1    typedef struct Node {
2    int number;          // 战士编号
3    struct Node* next;   // 下一个节点指针
4    } Node;
```

3.2 创建循环链表

用 for 循环每次开辟一个空间存放编号和后继地址，用尾插法将节点依次连接，最后将尾节点的后继地址设为头节点地址，并返回头节点地址。

```
1    Node* create(int n) {
2
3    Node* head = NULL;
4    Node* prev = NULL;
5
6    for (int i = 1; i <= n; ++i) {
7        Node* newNode = (Node*)malloc(sizeof(Node));
8        if (newNode == NULL) {
9            printf("Memory␣allocation␣failed!\n");
10           exit(1);
11       }
12       newNode->number = i;
13       newNode->next = NULL;
14
15       if (head == NULL) {
16           head = newNode;
17       } else {
18           prev->next = newNode;
19       }
20       prev = newNode;
21   }
22
23   // 将最后一个节点指向头节点，形成循环链表
24   prevnode = prev; //prevnode为全局变量，记录前驱节点
25   prev->next = head;
26   return head;
27   }
```

3.3 删除指定节点

删除指定节点时，首先判断头节点和指定节点地址，若有地址为空地址，则无法删除节点，返回 NULL。删除节点时有 2 种情况。若删除的节点是头节点，若链表中只剩下头节点，则直接删除，否则将前驱节点与头节点后继相连，释放头节点内存，将头节点重新设置为头节点的后继。若删除的节点不是头节点，则同样将前驱节点与被删除节点的后继节点连接，将删除节点的内存释放。最后都返回删除节点后继的地址，即下次报数的开头位置。

```
1     Node *deleteNode(Node *head, Node *nodeToDelete) {
2     if (head == NULL || nodeToDelete == NULL)
3         return NULL;
4
5     // 如果要删除的是头节点
6     if (nodeToDelete == head) {
7         if (head->next == head) { // 只有一个节点的情况
8             free(head);
9             return NULL;
10        }
11        prevnode->next = head->next;
12        free(head);
13        head = prevnode->next;
14        return prevnode->next;
15    }
16
17    // 如果要删除的不是头节点
18    prevnode->next = nodeToDelete->next;
19    free(nodeToDelete);
20    return prevnode->next;
21    }
```

3.4 判断加里森是否最后存活

该函数模拟报数和出列的过程，首先从头节点开始向后移动 x 个节点，找到开始报数的节点。然后开始轮回数数，若循环链表中的元素多于 1 个（即头节点的后继不为头节点），则向后移动 y 个节点。如果该节点的编号为 1，则直接返回 0，代表该情况已经不符合情况，否则调用函数删除该节点，继续循环。如果可以运行到循环结束，说明剩下的最后一个节点编号为 1，符合题目要求，返回 1。

需要注意的是，每次 while 循环都要更新前驱节点。

```
1     int isSurvivor(int n, int x, int y) {
2     Node *head = create(n);
3
4     // 找到起始节点
```

```
5       Node *current = head;
6       while (current->number != x) {
7           prevnode = current;
8           current = current->next;
9       }
10
11      // 开始轮回数数
12      while (head->next != head) {
13          for (int i = 1; i < y; ++i) {
14              prevnode = current;
15              current = current->next;
16          }
17          if (current->number == 1) { // 加里森被选中了
18              return 0;
19          }
20          current = deleteNode(head, current);
21      }
22
23      // 如果能执行到这一步，说明加里森成功活到了最后
24      return 1;
25  }
```

3.5 主函数设计

主函数中，首先输入 n，如果不符合题目要求会输出错误提醒。然后用双重 for 循环对 x,y 值遍历，循环内调用 IsSurvivor(n,x,y) 函数，若返回值为 1，则打印对应的 x,y 值。
该程序具有一定的可扩展性，通过更改 for 循环中的变量，可以探究在 x 固定的情况下，n,y 变量的合适输出，或者 y 固定的情况下，n,x 变量的合适输出。

```
1       int main() {
2       int n;
3       printf("Input␣n:␣");
4       scanf("%d", &n);
5       if(n<=0){
6           printf("Error␣data!\n");
7           return 0;
8       }
9       if(n==1){
10          printf("Is␣survivor\n");
11          return 0;
12      }
13      printf("x␣and␣y:␣\n");
```

```
14      for (int x = 1; x <= n; ++x) {
15          for (int y = 1; y <= n; ++y) {
16              if (isSurvivor(n, x, y)) {
17                  printf("x = %d, y = %d\n", x, y);
18              }
19          }
20      }
21
22      return 0;
23      }
```

4 调试分析报告

4.1 问题及解决方法

在编写程序的过程中，出现了空指针的问题，导致程序无法正常运行。解决方法是考虑所有情况，确保在所有情况下指针变量都能被赋值。
还出现了堆栈溢出的情况，原因是未能正确释放节点占用的内存。解决方法是删除节点时注意释放内存。

4.2 设计回顾与算法时空分析

该程序采用了模拟的方法，即模拟报数的具体流程来实现。
分析每个函数的时间复杂度，创建链表的时间复杂度为 $O(n)$, 删除指定节点的时间复杂度为 $O(1)$，isSurvivor(n,x,y) 函数的时间复杂度为 $O(n)$，主函数的时间复杂度为 $O(n^3)$。
空间复杂度上，本程序主要使用了一个长度为 n 的循环链表，每个节点由含有编号和后继地址的结构体组成。

4.3 改进设想

可以使用双向链表，在结构体节点中直接记录每个节点的前驱节点地址。虽然空间占用变大，但是使用较为方便，代码更简洁，同时可以避免使用全局变量。
或者，可以使用递推式（见附录 2）直接计算最后存活人的编号，而不使用模拟的方法，可以大量减小空间复杂度和代码量。下面是使用递推关系编写的程序。

```
1       int josephus(int n, int x, int y) {
2       int result = 0; // 最初假设编号为0的人是最后存活的人
3       for (int i = 2; i <= n; i++) {
4           result = (result + y) % i;
5       }
6       // 因为我们的编号是从1开始的，所以需要将结果加1
7       result = (result + x) % n;
8       if (result == 0) result = n; // 如果结果为0，则实际上是最后一个人
```

```
9       return result;
10      }
```

5 用户使用说明

将附录中的代码粘贴到 IDE 中编译运行，输入 n 值，将会连续输出满足题目要求的 x,y 值。

6 测试结果

6.1 固定 n 值测试

input: 0
output:

```
1       Error data!
```

input: 5
output:

```
1       x and y:
2   x = 1, y = 4
3   x = 2, y = 1
4   x = 3, y = 3
5   x = 4, y = 2
6   x = 5, y = 5
```

input: 10
output:

```
1       x and y:
2   x = 1, y = 8
3   x = 2, y = 1
4   x = 3, y = 7
5   x = 4, y = 10
6   x = 5, y = 9
7   x = 7, y = 2
8   x = 7, y = 4
9   x = 8, y = 3
10  x = 9, y = 5
11  x = 9, y = 6
```

6.2 固定 x 值测试

设置 n 的范围为 1 到 20。
input: x=4
output:

```
    n and y:
n = 4, y = 4
n = 4, y = 5
n = 4, y = 7
n = 4, y = 16
n = 4, y = 17
n = 4, y = 19
n = 5, y = 2
n = 5, y = 12
n = 5, y = 16
n = 5, y = 20
n = 6, y = 6
n = 6, y = 11
n = 6, y = 15
n = 7, y = 7
n = 7, y = 10
n = 7, y = 15
n = 7, y = 18
n = 8, y = 4
n = 9, y = 6
n = 9, y = 18
n = 10, y = 10
n = 11, y = 4
n = 11, y = 6
n = 11, y = 8
n = 11, y = 17
n = 12, y = 3
n = 13, y = 2
n = 13, y = 9
n = 13, y = 19
n = 14, y = 10
n = 15, y = 4
n = 16, y = 20
n = 17, y = 8
n = 17, y = 13
n = 18, y = 5
n = 18, y = 15
n = 19, y = 3
n = 19, y = 9
n = 19, y = 14
```

6.3 固定 y 值测试

n 值范围为 1 到 20
input: x=5
output:

```
    n and x:
n = 5, x = 5
n = 6, x = 1
n = 7, x = 3
n = 8, x = 7
n = 9, x = 3
n = 10, x = 9
n = 11, x = 5
n = 12, x = 1
n = 13, x = 9
n = 14, x = 5
n = 15, x = 1
n = 16, x = 12
n = 17, x = 8
n = 18, x = 4
n = 19, x = 19
n = 20, x = 15
```

A 附录 1：完整代码

```c
#include <stdio.h>
#include <stdlib.h>

// 定义链表节点
typedef struct Node {
    int number;          // 战士编号
    struct Node *next;   // 下一个节点指针
} Node;

Node *prevnode = NULL;

// 创建循环链表
Node *create(int n) {

    Node *head = NULL;
    Node *prev = NULL;
```

```
17
18      for (int i = 1; i <= n; ++i) {
19          Node *newNode = (Node *) malloc(sizeof(Node));
20          if (newNode == NULL) {
21              printf("Memory allocation failed!\n");
22              exit(1);
23          }
24          newNode->number = i;
25          newNode->next = NULL;
26
27          if (head == NULL) {
28              head = newNode;
29          } else {
30              prev->next = newNode;
31          }
32          prev = newNode;
33      }
34
35      // 将最后一个节点指向头节点，形成循环链表
36      prevnode = prev;
37      prev->next = head;
38
39      return head;
40  }
41
42  // 删除指定节点
43  Node *deleteNode(Node *head, Node *nodeToDelete) {
44      if (head == NULL || nodeToDelete == NULL)
45          return NULL;
46
47      // 如果要删除的是头节点
48      if (nodeToDelete == head) {
49          if (head->next == head) { // 只有一个节点的情况
50              free(head);
51              return NULL;
52          }
53          prevnode->next = head->next;
54          free(head);
55          head = prevnode->next;
56          return prevnode->next;
57      }
```

```
58
59      // 如果要删除的不是头节点
60      prevnode->next = nodeToDelete->next;
61      free(nodeToDelete);
62      return prevnode->next;
63  }
64
65  // 检查某个 x,y 组合是否使加里森活到最后
66  int isSurvivor(int n, int x, int y) {
67      Node *head = create(n);
68
69      // 找到起始节点
70      Node *current = head;
71      while (current->number != x) {
72          prevnode = current;
73          current = current->next;
74      }
75
76      // 开始轮回数数
77      while (head->next != head) {
78          for (int i = 1; i < y; ++i) {
79              prevnode = current;
80              current = current->next;
81          }
82          if (current->number == 1) { // 加里森被选中了
83              return 0;
84          }
85          current = deleteNode(head, current);
86      }
87
88      // 如果能执行到这一步，说明加里森成功活到了最后
89      return 1;
90  }
91
92  // 主函数，输出所有使加里森活到最后的 x 和 y 值
93  int main() {
94      int n;
95      printf("Input n: ");
96      scanf("%d", &n);
97      if (n <= 0) {
98          printf("Error data!\n");
```

```
99          return 0;
100     }
101     if (n == 1) {
102         printf("Is survivor\n");
103         return 0;
104     }
105     printf("x and y: \n");
106     for (int x = 1; x <= n; ++x) {
107         for (int y = 1; y <= n; ++y) {
108             if (isSurvivor(n, x, y)) {
109                 printf("x = %d, y = %d\n", x, y);
110             }
111         }
112     }
113
114 return 0;
115 }
```

B 附录 2：递推式

$$f(n,m) = \begin{cases} 0 & n = 1 \\ [f(n-1,m)+m]\%n & n > 1 \end{cases}$$