题目描述

100个人围成一圈，每个人有一个编码，编码从1开始到100。

他们从1开始报出对编数，报到3的倍数的人退出圈子，报到不是3的倍数的人还在圈子里，最后留在圈子的人编号是多少？

请编写程序模拟这个过程，并输出最后留在圈子里的人的编号。

输入描述

输入一个整数参数 M

输出描述

如果输入参数M小于等于1或者大于等于100，输出字符串"ERROR!"；

否则按照题目要求计算出最后一个留在圈子里的人的编号，并输出该编号。

用例

输入	3
输出	10.97
说明	输入3时，最后剩下两个人。

输入	0
输出	34.85.97
说明	输入0时，最后剩下三个人。

题目解析

本题最经典的约瑟夫环问题了，可以用传统环形数据结构。

解法一： 循环链表[1]

定义一个循环链表，头节点value=1，尾节点value=100，头.prev = 尾，尾.next = 头。
这样可以，顺利造出了一个循环链表。
我们还需要为循环链表增加删除功能，用来删去报数节点，继续往复节点的操作。
初始，我们定位要删除第一个报数位置的人，从1开始。
这样的话，我们从第三个node开始报数，这时报数cur.val=3，

- 处理cur.val=3，说明该人报数成功，删去cur，下一个报数节点应为该删除节点的下一个节点，并且重置报数cur=1。
- 处理cur.val=1，说明该报数第下一节点，即cur = cur.next，并且报数cur++

按上述逻辑，重复操作到为cur = cur.next，结束。
那时，head.val =cur, ++cur 循环出循环体时，剩下一个节点值，就是结果。

Java算法源码

```
import java.util.Scanner;
import java.util.StringBuilder;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int m = sc.nextInt();
    System.out.println(getResult(m));
  }

  public static String getResult(int m) {
    if (m <= 1 || m >= 100) return "ERROR!";

    CyclicDoubleList list = new CyclicDoubleList();
    for (int i = 1; i <= 100; i++) list.append(i);

    int idx = 1;
    Node cur = list.head;

    while (list.size >= m) {
      if (idx == 3) {
        idx = 1;
        cur = list.remove(cur);
      } else {
        idx++;
        cur = cur.next;
      }
    }

    return list.toString();
  }
}

class Node {
  int val;
  Node prev;
  Node next;

  public Node(int val, Node prev, Node next) {
    this.val = val;
    this.prev = prev;
    this.next = next;
  }
}

class CyclicDoubleList {
  Node head;
  Node tail;
  int size = 0;

  public void append(int val) {
    Node node = new Node(val, null, null);

    if (this.size == 0) {
      this.tail.next = node;
      node.prev = this.tail;
      this.head = node;
    } else {
      this.head = node;
      this.tail = node;
    }
    this.head.prev = this.tail;
    this.tail.next = this.head;
    this.size++;
  }

  public Node remove(Node cur) {
    Node pre = cur.prev;
    Node nxt = cur.next;

    pre.next = nxt;
    nxt.prev = pre;

    cur.next = cur.prev = null;

    if (this.head == cur) {
      this.head = nxt;
    }

    if (this.tail == cur) {
      this.tail = pre;
    }
    this.size--;
    return nxt;
  }

  @Override
  public String toString() {
    StringBuilder sj = new StringBuilder("[");

    Node cur = this.head;
    for (int i = 0; i < this.size; i++) {
      sj.add(cur.val + "");
      cur = cur.next;
    }

    return sj.toStringBuilder();
  }
}
```

JS算法源码

```
/* eslint-disable-line */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const m = line - 0;
  console.log(getResult(m));
});

function getResult(m) {
  if (m <= 1 || m >= 100) {
    return console.log("ERROR!");
  }

  const list = new CyclicDoubleList();
  for (let i = 1; i <= 100; i++) list.append(i);

  let idx = 1;
  let cur = list.head;

  while (list.size >= m) {
    if (idx === 3) {
      idx = 1;
      cur = list.remove(cur);
    } else {
      idx++;
      cur = cur.next;
    }
  }

  return list.toString();
}

/* 节点类型 */
class Node {
  constructor(val) {
    this.val = val;
    this.next = null;
    this.prev = null;
  }
}

/* 循环双向表 */
class CyclicDoubleList {
  constructor() {
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  append(val) {
    const node = new Node(val);

    if (this.size) {
      this.tail.next = node;
      node.prev = this.tail;
      this.tail = node;
    } else {
      this.head = node
      this.tail = node;
    }

    this.head.prev = this.tail;
    this.tail.next = this.head;
    this.size++;
  }

  remove(cur) {
    const pre = cur.prev;
    const nxt = cur.next;

    pre.next = nxt;
    nxt.prev = pre;

    cur.next = cur.prev = null;

    if (this.head === cur) {
      this.head = nxt;
    }

    if (this.tail === cur) {
      this.tail = pre;
    }

    this.size--;

    return nxt;
  }

  toStringJ() {
    const arr = [];
    let cur = this.head;

    for (let i = 0; i < this.size; i++) {
      arr.push(cur.val);
      cur = cur.next;
    }

    return arr.join();
  }
}
```

Python算法源码

```
# 节点类型
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None
        self.prev = None

# 循环双向表
class CyclicDoubleList:
    def __init__(self):
        self.head = None
        self.tail = None
        self.size = 0

    def append(self, val):
        node = Node(val)

        if self.size > 0:
            self.tail.next = node
            node.prev = self.tail
            self.tail = node
        else:
            self.head = node
            self.tail = node

        self.head.prev = self.tail
        self.tail.next = self.head
        self.size += 1

    def remove(self, cur):
        pre = cur.prev
        nxt = cur.next

        pre.next = nxt
        nxt.prev = pre

        cur.next = cur.prev = None

        if self.head == cur:
            self.head = nxt

        if self.tail == cur:
            self.tail = pre

        self.size -= 1

        return nxt

    def __str__(self):
        arr = []
        cur = self.head

        for i in range(self.size):
            arr.append(str(cur.val))
            cur = cur.next

        return ",".join(arr)

# 输入数据
m = int(input())

# 输出结果
def getResult(m):
    if m <= 1 or m >= 100:
        return "ERROR!"

    cyclist = CyclicDoubleList()
    for i in range(1, 101):
        cyclist.append(i)

    idx = 1
    cur = cyclist.head

    while cyclist.size >= m:
        if idx == 3:
            idx = 1
            cur = cyclist.remove(cur)
        else:
            idx += 1
            cur = cur.next

    return str(cyclist)

# 输出结果
print(getResult())
```