# 【华为OD机考 统一考试机试C卷】求幸存数之和（C++ Java JavaScript Python）

## 华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试( B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

## 题目描述

给一个正整数数列 nums，一个跳数 jump，及幸存数量 left。
运算过程为：从索引0的位置开始向后跳，中间跳过 J 个数字，命中索引为 J+1 的数字，该数被敲出，并从该点起跳，以此类推，直到幸存 left 个数为止，然后返回幸存数之和。

约束：

- 0是第一个起跳点
- 起跳点和命中点之间间隔 jump 个数字，已被敲出的数字不计入在内。
- 跳到末尾时无缝从头开始（循环查找），并可以多次循环。
- 若起始时 left > len(nums) 则无需跳数处理过程。

方法设计：

```
1 * @param nums 正整数数列，长度范围 [1, 10000]
2 * @param jump 跳数，范围 [1, 10000]
3
```

```
 * @param left 幸存数量，范围 [0, 10000]
 * @return 幸存数之和
int sumOfLeft(int[] nums, int jump, int left){

}
```

## 输入描述

第一行输入正整数数列

第二行输入跳数

第三行输入幸存数量

## 输出描述

输出幸存数之和

## 用例

| | |
|---|---|
| **输入** | **1,2,3,4,5,6,7,8,9**<br>**4**<br>**3** |
| 输出 | 13 |
| 说明 | 从1（索引为0）开始起跳，中间跳过 4 个数字，因此依次删除 6,2,8,5,4,7。剩余1,3,9，返回和为13 |

## 解题思路

本题考试时为Lettoce模式,无需自己获取输入数据。

本题主要是模拟操作，按照每次跳数的位置，从数列中删掉跳到的数组，直到剩余幸存数量的数字。

## C++

```
#include <iostream>
#include <list>
#include <numeric>

```

```cpp
#include <sstream>
#include <string>
#include <vector>
#include <numeric>
using namespace std;
// 计算幸存数之和
int sumOfLeft(int nums[], int jump, int left, int length) {
    // 如果幸存数量大于等于数组长度，则直接返回数组元素之和
    if (left >= length) {
        return accumulate(nums, nums + length, 0);
    }

    // 使用vector存储数组元素，方便删除操作
    vector<int> lst(nums, nums + length);

    // 初始化起跳点索引为0
    int index = 0;
    // 当列表大小大于幸存数量时，执行删除操作
    while (lst.size() > left) {
        // 计算下一个要删除元素的索引
        index = (index + jump + 1) % lst.size();
        // 删除计算出的索引处的元素
        lst.erase(lst.begin() + index);
        // 由于删除元素后，列表会缩短，下一个起跳点应当向前移动一位
        index = index - 1  ;
    }

    // 计算并返回剩余元素之和
    return accumulate(lst.begin(), lst.end(), 0);
}

int main() {
    string line;
    vector<int> nums;
    int jump, left;

    // 读取一行输入，按逗号分割，转换为整数数组
    getline(cin, line);
    stringstream ss(line);
    string num;
```

```
    while (getline(ss, num, ',')) {
        nums.push_back(stoi(num));
    }

    // 读取跳数
    cin >> jump;
    // 读取幸存数量
    cin >> left;

    // 输出幸存数之和
    cout << sumOfLeft(&nums[0], jump, left, nums.size()) << endl;

    return 0;
}
```

## Java

```
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // 读取一行输入，按逗号分割，转换为整数数组
        int[] nums = Arrays.stream(sc.nextLine().split(",")).mapToInt(Integer::parseInt).toArray();
        // 读取跳数
        int jump = Integer.parseInt(sc.nextLine());
        // 读取幸存数量
        int left = Integer.parseInt(sc.nextLine());

        // 输出幸存数之和
        System.out.println(sumOfLeft(nums, jump, left));
    }

    public static int sumOfLeft(int[] nums, int jump, int left) {
        // 如果幸存数量大于等于数组长度，则直接返回数组元素之和
        if (left >= nums.length) {
            return Arrays.stream(nums).sum();
        }
```

```
        // 使用LinkedList存储数组元素，方便删除操作
        LinkedList<Integer> list = new LinkedList<>();
        for (int num : nums) {
            list.add(num);
        }

        // 初始化起跳点索引为0
        int index = 0;
        // 当列表大小大于幸存数量时，执行删除操作
        while (list.size() > left) {
            // 计算下一个要删除元素的索引
            index = index + jump + 1;
            // 为了实现循环跳跃，索引可能会超出列表大小，因此取模
            index = index % list.size();
            // 删除计算出的索引处的元素
            list.remove(index);
            // 由于删除元素后，列表会缩短，下一个起跳点应当向前移动一位
            index = index - 1;
        }

        // 计算并返回剩余元素之和
        return list.stream().mapToInt(Integer::intValue).sum();
    }
}
```

## javaScript

```
// 导入必要的库
const readline = require('readline');

// 创建readline接口实例
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// 读取输入
rl.on('line', (line) => {
  // 根据输入行数，分别处理
```

```
  if (!this.nums) {
    // 读取一行输入，按逗号分割，转换为整数数组
    this.nums = line.split(',').map(Number);
  } else if (!this.jump) {
    // 读取跳数
    this.jump = Number(line);
  } else if (!this.left) {
    // 读取幸存数量
    this.left = Number(line);
    // 输出幸存数之和
    console.log(sumOfLeft(this.nums, this.jump, this.left));
    rl.close();
  }
});

// 计算幸存数之和
function sumOfLeft(nums, jump, left) {
  // 如果幸存数量大于等于数组长度，则直接返回数组元素之和
  if (left >= nums.length) {
    return nums.reduce((acc, val) => acc + val, 0);
  }

  // 使用数组存储元素，方便删除操作
  let list = nums.slice();

  // 初始化起跳点索引为0
  let index = 0;
  // 当列表大小大于幸存数量时，执行删除操作
  while (list.length > left) {
    // 计算下一个要删除元素的索引
    index = (index + jump + 1) % list.length;
    // 删除计算出的索引处的元素
    list.splice(index, 1);
    // 由于删除元素后，列表会缩短，下一个起跳点应当向前移动一位
    index = index - 1  ;
  }

  // 计算并返回剩余元素之和
  return list.reduce((acc, val) => acc + val, 0);
}
```

## Python

```python
def sum_of_left(nums, jump, left):
    # 如果幸存数量大于等于数组长度，则直接返回数组元素之和
    if left >= len(nums):
        return sum(nums)

    # 使用列表存储数组元素，方便删除操作
    lst = nums[:]

    # 初始化起跳点索引为0
    index = 0
    # 当列表大小大于幸存数量时，执行删除操作
    while len(lst) > left:
        # 计算下一个要删除元素的索引
        index = (index + jump + 1) % len(lst)
        # 删除计算出的索引处的元素
        del lst[index]
        # 由于删除元素后，列表会缩短，下一个起跳点应当向前移动一位
        index = index - 1

    # 计算并返回剩余元素之和
    return sum(lst)

# 读取一行输入，按逗号分割，转换为整数数组
nums = list(map(int, input().split(',')))
# 读取跳数
jump = int(input())
# 读取幸存数量
left = int(input())

# 输出幸存数之和
print(sum_of_left(nums, jump, left))
```

## 完整用例

### 用例1

1,2,3,4,5,6,7,8,9,10
2
4

### 用例2

1,2,3,4,5
6
2

### 用例3

1,2,3,4,5
1
5

### 用例4

1,2,3,4,5,6,7,8,9,10
3
0

### 用例5

1
1
1

### 用例6

1,2,3,4,5,6,7,8,9,10
1
3

### 用例7

1,2,3,4,5,6,7,8,9,10
10

5

### 用例8

2,4,6,8,10,12
3
2

### 用例9

1,2
1
1

### 用例10

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
14
3

### 文章目录

用例5
用例6
用例7
用例8
用例9
用例10