# 题目

给你两个整数数组 nums 和 index。你需要按照以下规则创建目标数组：

目标数组 target 最初为空。

按从左到右的顺序依次读取 nums[i] 和 index[i]，在 target 数组中的下标 index[i] 处插入值 nums[i] 。

重复上一步，直到在 nums 和 index 中都没有要读取的元素。

请你返回目标数组。

题目保证数字插入位置总是存在。

示例 1：

输入：nums = [0,1,2,3,4], index = [0,1,2,2,1]

输出：[0,4,1,3,2]

解释：

nums index target

0 0 [0]

1 1 [0,1]

2 2 [0,1,2]

3 2 [0,1,3,2]

4 1 [0,4,1,3,2]

示例 2：

输入：nums = [1,2,3,4,0], index = [0,1,2,3,0]

输出：[0,1,2,3,4]

解释：

nums index target

1 0 [1]

2 1 [1,2]

3 2 [1,2,3]

4 3 [1,2,3,4]

0 0 [0,1,2,3,4]

示例 3：

输入：nums = [1], index = [0]

输出：[1]

提示：

1 <= nums.length, index.length <= 100

nums.length == index.length

0 <= nums[i] <= 100

0 <= index[i] <= i

# 分析

## 方法一：模拟

思路：

这个解决方案使用 std::vector 的 insert 方法来在目标数组的指定位置插入元素。它遍历 nums 和 index 数组，对于每个元素 nums[i]，它将其插入到目标数组 target 的 index[i] 位置。最后，返回构建好的目标数组 target。

请注意，这种方法的时间复杂度较高，因为每次插入操作都可能导致目标数组中的元素进行移动。在最坏的情况下，时间复杂度为 O(n^2)，其中 n 是 nums 数组的长度。如果输入规模较大，可能需要考虑优化算法以提高效率。

另外，根据题目提示，index[i] 的值满足 0 <= index[i] <= i，这意味着每次插入的位置都是在当前目标数组的末尾或之前，因此可以使用 push\_back 和 rotate 的组合操作来优化插入过程，避免每次插入都导致整个数组的移动。但考虑到这里只是给出了一个简单的解决方案，没有进行额外的优化。

代码：

class Solution {

public:

std::vector<int> createTargetArray(std::vector<int>& nums, std::vector<int>& index) {

std::vector<int> target;

int n = nums.size();

for (int i = 0; i < n; i++) {

// 在目标数组的指定位置插入元素

target.insert(target.begin() + index[i], nums[i]);

}

return target;

}

};