Question **1**

Correct

Marked out of 1.00

Flag question

Given an array of integers, reverse the given array in place using an index and loop rather than a built-in function.

Example

arr = [1, 3, 2, 4, 5]

Return the array *[5, 4, 2, 3, 1]* which is the reverse of the input array.

Function Description

Complete the function *reverseArray* in the editor below.

reverseArray has the following parameter(s):

int arr[n]: an array of integers

Return

int[n]: the array in reverse order

Constraints

1 ≤ n ≤ 100

0 < arr[i] ≤ 100

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *arr*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer, *arr[i]*.

Sample Case 0

Sample Input For Custom Testing

5

1

3

2

4

5

Sample Output

5

4

2

3

1

Explanation

The input array is [1, 3, 2, 4, 5], so the reverse of the input array is [5, 4, 2, 3, 1].

Sample Case 1

Sample Input For Custom Testing

4

17

10

21

45

Sample Output

45

21

10

17

Explanation

The input array is [17, 10, 21, 45], so the reverse of the input array is [45, 21, 10, 17].

Answer: (penalty regime: 0 %)

Reset answer

```
29          }
30  *
31  *      return a;
32  * }
33  *
34  */
35  #include<stdio.h>
36  #include<stdlib.h>
37  int* reverseArray(int arr_count
38      int* result =(int*)malloc(a
39
40      if(result ==NULL){
41          return NULL;
42      }
43      for(int i=0;i<arr_count;i++
44      {
45          result[i]=arr[arr_count
46      }
47      *result_count =arr_count;
48      return result;
49
50
51  }
52
```

	Test

✓	`int arr[] = {1, 3, 2, 4, 5};`
	`int result_count;`
	`int* result = reverseArray(5, arr,`
	`for (int i = 0; i < result_count;`
	`printf("%d\n", *(result +`

Passed all tests! ✓

Question **2**

Correct

Marked out of 1.00

Flag question

An automated cutting machine is used to cut rods into segments. The cutting machine can only hold a rod of *minLength* or more, and it can only make one cut at a time. Given the array *lengths[]* representing the desired lengths of each segment, determine if it is possible to make the necessary cuts using this machine. The rod is marked into lengths already, in the order given.

Example

n = 3

lengths = [4, 3, 2]

minLength = 7

The rod is initially *sum(lengths) = 4 + 3 + 2 = 9* units long. First cut off the segment of length *4 + 3 = 7* leaving a rod *9 - 7 = 2*. Then check that the length *7* rod can be cut into segments of lengths *4* and *3*. Since *7* is greater than or equal to *minLength = 7*, the final cut can be made. Return *"Possible"*.

Example

n = 3

lengths = [4, 2, 3]

minLength = 7

The rod is initially *sum(lengths) = 4 + 2 + 3 = 9* units long. In this case, the initial cut can be of length *4* or *4 + 2 = 6*. Regardless of the length of the first cut, the remaining piece will be shorter than *minLength*. Because *n - 1 = 2* cuts cannot be made, the answer is *"Impossible"*.

Function Description

Complete the function *cutThemAll* in the editor below.

cutThemAll has the following parameter(s):

int lengths[n]: the lengths of the segments, in order

int minLength: the minimum length the machine can accept

Returns

string: *"Possible"* if all *n-1* cuts can be made. Otherwise, return the string *"Impossible"*.

Constraints

- $2 \le n \le 10^5$
- $1 \le t \le 10^9$
- $1 \le lengths[i] \le 10^9$
- *The sum of the elements of lengths equals the uncut rod length.*

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *lengths*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i*

```
 * The function is expected to
 * The function accepts followi
 *  1. LONG_INTEGER_ARRAY lengt
 *  2. LONG_INTEGER minLength
 */

/*
 * To return the string from th
 *
 * For example,
 * char* return_string_using_st
 *     static char s[] = "stati
 *
 *     return s;
 * }
 *
 * char* return_string_using_dy
 *     char* s = malloc(100 * s
 *
 *     s = "dynamic allocation
 *
 *     return s;
 * }
 *
 */
#include<stdio.h>

char* cutThemAll(int lengths_co
long t=0,i=1;
for (int i=0;i<=lengths_count-1
    t+=lengths[i];
}
do{
    if(t-lengths[lengths_count-
        return "Impossible";
    }
    i++;
}
while(i<lengths_count-i);
return "Possible";

}
```

	Test
✓	long lengths[] = {3, 5, 4, 3}; printf("%s", cutThemAll(4, lengths
✓	long lengths[] = {5, 6, 2}; printf("%s", cutThemAll(3, lengths

Passed all tests! ✓