Your attempts

Attempt 1

Status	Finished
Started	Monday, 13 January 2025, 11:57 AM
Completed	Monday, 13 January 2025, 12:08 PM
Duration	10 mins 25 secs

Review

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

$1 \le n \le 100$

$0 < arr[i] \le 100$

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 $0 \le i < n$) contains an integer, *arr[i]*.

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

```c
/*
 * Complete the 'reverseArray' f
 *
 * The function is expected to r
 * The function accepts INTEGER_
 */

/*
 * To return the integer array f
 *     - Store the size of the a
 *     - Allocate the array sta
 *
 * For example,
 * int* return_integer_array_us
 *     *result_count = 5;
 *
 *     static int a[5] = {1, 2,
 *
 *     return a;
 * }
 *
 * int* return_integer_array_us
 *     *result_count = 5;
 *
 *     int *a = malloc(5 * size
 *
 *     for (int i = 0; i < 5; i+
 *         *(a + i) = i + 1;
 *     }
 *
 *     return a;
 * }
 *
 */
#include<stdio.h>
#include<stdlib.h>
int* reverseArray(int arr_count
    int* result =(int*)malloc(ar
    if(result==NULL){
        return NULL;
```

```
41      }
42      for(int i=0;i<arr_count;i++
43      {
44          result[i]=arr[arr_count
45      }
46      *result_count=arr_count;
47      return result;
48  }
49
```

	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! ✓

	Expected	Got	
arr, &result_count); nt; i++) t + i));	5 4 2 3 1	5 4 2 3 1	✓

Passed all tests! ✓

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 $0 \le i < n$) contains an integer, *lengths[i]*.

The next line contains an integer, *minLength*, the minimum length accepted by the machine.

Sample Case 0

Sample Input For Custom Testing

STDIN Function

----- --------

4 → lengths[] size n = 4

3 → lengths[] = [3, 5, 4, 3]

5

4

3

9 → minLength= 9

Sample Output

Possible

Explanation

The uncut rod is *3 + 5 + 4 + 3 = 15* units long. Cut the rod into lengths of *3 + 5 + 4 = 12* and *3*. Then cut the *12* unit piece into lengths *3* and *5 + 4 = 9*. The remaining segment is *5 + 4 = 9* units and that is long enough to make the final cut.

Sample Case 1

Sample Input For Custom Testing

STDIN Function

----- --------

3 → lengths[] size n = 3

5 → lengths[] = [5, 6, 2]

6

2

12 → minLength= 12

Sample Output

Impossible

Explanation

The uncut rod is *5 + 6 + 2 = 13* units long. After making either cut, the rod will be too short to make the second cut.

```
/*
 * Complete the 'cutThemAll' fu
 *
 * 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;i+
    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! ✓

	Expected	Got	
; ngths, 9))	Possible	Possible	✓
 ngths, 12))	Impossible	Impossible	✓

Passed all tests! ✓