GE23131-Programming Using C-2024

Status	Finished
Started	Wednesday, 15 January 2025, 7:30 PM
Completed	Wednesday, 15 January 2025, 8:01 PM
Duration	30 mins 46 secs

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

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

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].

```c
/*
 * Complete the 'reverseArra
 *
 * The function is expected
 * The function accepts INTE
 */

/*
 * To return the integer arr
 *     - Store the size of t
 *     - Allocate the array
 *
 * For example,
 * int* return_integer_array
 *     *result_count = 5;
 *
 *     static int a[5] = {1,
 *
 *     return a;
 * }
 *
 * int* return_integer_array
 *     *result_count = 5;
 *
 *     int *a = malloc(5 * s
 *
 *     for (int i = 0; i < 5
 *         *(a + i) = i + 1;
 *     }
 *
 *     return a;
 * }
 *
 */
int* reverseArray(int arr_co
{
    *result_count=arr_count;
    for(int i=0;i<arr_count/
    {
```

```
 */
int* reverseArray(int arr_co
{
    *result_count=arr_count;
    for(int i=0;i<arr_count/
    {
        int temp=arr[i];
        arr[i]=arr[arr_count
        arr[arr_count-i-1]=t
    }
    return arr;
}

```

	Test

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

Passed all tests! ✓

Question **2**

Correct

Marked out of 1.00

Flag question

```
32
33
34
35  int *arr, int *result_count)
36
37
38  +)
39
40
41  ];
42
43
44
45
46
```

	Expected	Got	
&result_count); i++) i));	5 4 2 3 1	5 4 2 3 1	✓

Passed all tests! ✓

Question **2**

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):

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

- $2 \leq n \leq 10^5$
- $1 \leq t \leq 10^9$
- $1 \leq lengths[i] \leq 10^9$
- *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

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.

```c
/*
 * Complete the 'cutThemAll'
 *
 * The function is expected
 * The function accepts foll
 *   1. LONG_INTEGER_ARRAY le
 *   2. LONG_INTEGER minLengt
 */

/*
 * To return the string from
 *
 * For example,
 * char* return_string_using
 *      static char s[] = "st
 *
 *      return s;
 * }
 *
 * char* return_string_using
 *      char* s = malloc(100
 *
 *      s = "dynamic allocati
 *
 *      return s;
 * }
 *
 */
char* cutThemAll(int lengths
    long t=0,i=1;
    for(int i=0;i<=lengths_c
    {
        t+=lengths[i];
    }
    do
```

```
19
20  string_using_dynamic_allocati
21   malloc(100 * sizeof(char));
22
23  mic allocation of string";
24
25
26
27
28
29  (int lengths_count, long *len
30  ;
31  i<=lengths_count-1;i++)
32
33  hs[i];
34
35
36
37  gths[lengths_count-i-1]<minLe
38
39  rn "Impossible";
40
41
42  ngths_count-1);
43  ';
44
45
46
47
```

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

```
27
28
29 ngths, long minLength){
30
31
32
33
34
35
36
37  ength)
38
39
40
41
42
43
44
45
46
47
```

	Expected	Got	
ths, 9))	Possible	Possible	✓
ths, 12))	Impossible	Impossible	✓

Passed all tests! ✓