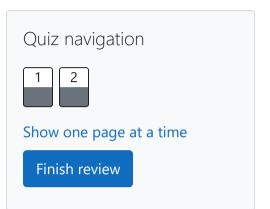
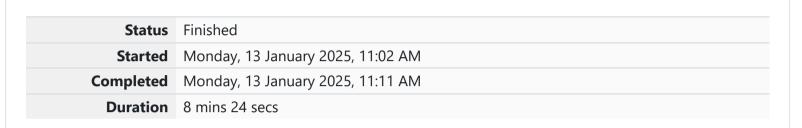
# GE23131-Programming Using C-2024





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**

 $1 \le n \le 100$ 

 $0 < arr[i] \le 100$ 

# **Input Format For Custom Testing**

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

# **Sample Input For Custom Testing** 5 3 2 5 **Sample Output** 5 4 **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

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

```
1 | /*
     * Complete the 'reverseArray' function below.
 3
     * The function is expected to return an INTEGER ARRAY.
 4
     * The function accepts INTEGER ARRAY arr as parameter.
 5
 6
 7
 8
     * To return the integer array from the function, you should:
 9
           - Store the size of the array to be returned in the resul
10
           - Allocate the array statically or dynamically
11
12
     * For example,
13
     * int* return integer array using static allocation(int* result
14
15
           *result count = 5;
16
           static int a[5] = \{1, 2, 3, 4, 5\};
17
18
19
           return a;
20
21
     * int* return integer array using dynamic allocation(int* resul
22
           *result count = 5;
23
24
25
           int *a = malloc(5 * sizeof(int));
26
           for (int i = 0; i < 5; i++) {
27 🔻
               *(a + i) = i + 1;
28
29
30
31
           return a;
```

```
#include<stdio.h>
35
     #include<stdlib.h>
36
    int* reverseArray(int arr count, int *arr, int *result count) {
37 1
        int* result =(int*)malloc(arr count * sizeof(int));
38
39
        if(result ==NULL){
40
            return NULL;
41
42
        for (int i=0;i<arr_count;i++)</pre>
43
44
            result[i]=arr[arr_count-i-1];
45
46
        *result count =arr count;
47
        return result;
48
49
50
51
52
```

	Test	Expected	Got	
<b>~</b>	int arr[] = {1, 3, 2, 4, 5};	5	5	~
	<pre>int result_count;</pre>	4	4	
	<pre>int* result = reverseArray(5, arr,</pre>	2	2	
	<pre>&amp;result_count);</pre>	3	3	
	for (int i = 0; i < result_count; i++)	1	1	
	<pre>printf("%d\n", *(result + i));</pre>			

Passed all tests! <

Question  ${\bf 2}$ 

Correct

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

r riag question

# **Example**

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.

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

- $\cdot \qquad 2 \le n \le 10^5$
- $\cdot 1 \le t \le 10^9$
- $1 \le lengths[i] \le 10^9$
- 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

```
STDIN Function
    \rightarrow lengths[] size n = 4
    \rightarrow lengths[] = [3, 5, 4, 3]
5
4
3
    → 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 = 15
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
```

 $\rightarrow$  lengths[] size n = 3

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.

**Answer:** (penalty regime: 0 %)

#### Reset answer

```
* Complete the 'cutThemAll' function below.
 3
     * The function is expected to return a STRING.
     * The function accepts following parameters:
     * 1. LONG INTEGER ARRAY lengths
     * 2. LONG INTEGER minLength
 8
 9
10 •
     * To return the string from the function, you should either do
11
12
     * For example,
13
     * char* return_string_using_static_allocation() {
14
           static char s[] = "static allocation of string";
15
16
17
           return s:
```

```
21
            char* s = malloc(100 * sizeof(char));
22
            s = "dynamic allocation of string";
23
24
25
            return s;
26
27
28
29
     #include<stdio.h>
    char* cutThemAll(int lengths_count, long *lengths, long minLengt
    long t=0,i=1;
31
32 * for(int i=0;i<=lengths_count-1;i++){</pre>
        t+=lengths[i];
33
34
35 ▼
    do{
        if(t-lengths[lengths_count-1]<minLength){</pre>
36 ▼
             return "Impossible";
37
38
39
        i++;
    }while(i<lengths_count-i);</pre>
    return "Possible";
41
42
43
44
45
46
47
48
49
50
51
52
```

	Test	Expected	Got	
~	<pre>long lengths[] = {3, 5, 4, 3}; printf("%s", cutThemAll(4, lengths, 9))</pre>	Possible	Possible	<b>~</b>

