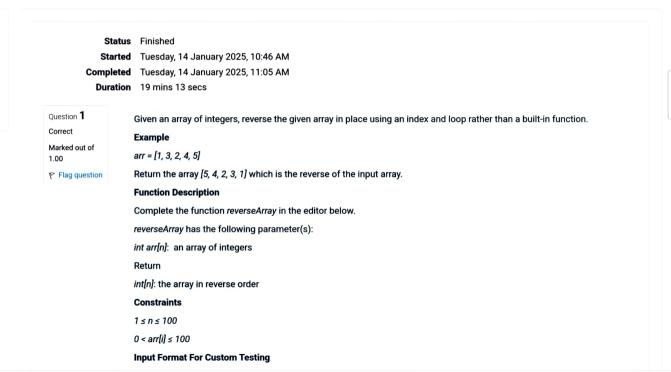
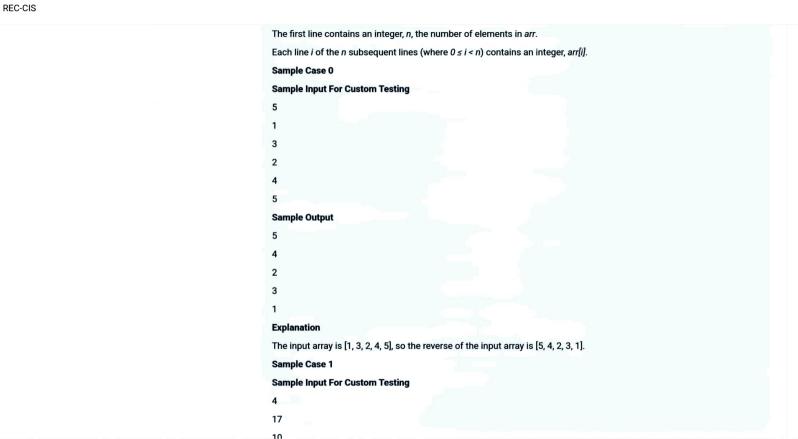
# GE23131-Programming Using C-2024







**REC-CIS 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 \* Complete the 'reverseArray' function below. \* The function is expected to return an INTEGER\_ARRAY. \* The function accepts INTEGER\_ARRAY arr as parameter.

```
Reset answer
  1 - /*
     * Complete the 'reverseArray' function below.
 3
     * The function is expected to return an INTEGER_ARRAY.
      * The function accepts INTEGER_ARRAY arr as parameter.
 8 ₩
     * To return the integer array from the function, you should:
            - Store the size of the array to be returned in the result_count variable
10
            - Allocate the array statically or dynamically
11
12
13
     * For example,
      * int* return_integer_array_using_static_allocation(int* result_count) {
14 ▼
            *result count = 5;
15
16
            static int a[5] = \{1, 2, 3, 4, 5\};
17
18
            return a;
19
20
21
      * int* return_integer_array_using_dynamic_allocation(int* result_count) {
22 🔻
            *result count = 5;
23
24
            int *a = malloc(5 * sizeof(int));
25
 26
            for (int i = 0; i < 5; i++) {
27 ▼
                *(a + i) = i + 1;
 28
 29
30
31
            return a;
32
      * 1
33
 34
```

35 #include<stdio.h>

```
33
34
    #include<stdio.h>
   #include<stdlib.h>
    int* reverseArray(int arr_count, int *arr, int *result_count)
37
38 ▼ {
39
        int* result=(int*)malloc(arr_count * sizeof(int));
40
        if(result==NULL)
41 ▼
            return NULL:
42
43
        for(int i=0;i<arr_count;i++)</pre>
44
45 ▼
            result[i]=arr[arr_count-i-1];
46
47
48
        *result_count=arr_count;
49
        return result;
50
51 }
52
```

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

Ouestion 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

Example

n = 3

n = 3lengths = [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".

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

## Complete the function cutThemAll in the editor below.

**Function Description** 

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 ≤ n ≤ 10<sup>5</sup>
- $1 \le t \le 10^9$
- 1 ≤ lengths[i] ≤ 10<sup>9</sup>
- The sum of the elements of lengths equals the uncut rod length.

### Input Format For Custom Testing



#### 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

 $\rightarrow$  lengths[] = [5, 6, 2]

6

2

12 → minLength= 12

Sample Output

## Sample Output

Impossible

## Explanation

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
10 ▼
11
     * To return the string from the function, you should either do static allocation or dyna
12
13
     * For example,
14 ▼
       char* return_string_using_static_allocation() {
15
           static char s[] = "static allocation of string";
16
17
           return s;
12
```

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. \* The function is expected to return a STRING. \* The function accepts following parameters: \* 1. LONG INTEGER ARRAY lengths 2. LONG INTEGER minLength 10 ▼ 11 \* To return the string from the function, you should either do static allocation or dyna 12 13 \* For example. \* char\* return\_string\_using\_static\_allocation() { 15 static char s[] = "static allocation of string"; 16 17 return s: 18 19 20 ▼ \* char\* return\_string\_using\_dynamic\_allocation() { 21 char\* s = malloc(100 \* sizeof(char)): s = "dynamic allocation of string"; 23 24 return s; 26 27 28 #include<stdio.h> 30 v char\* cutThemAll(int lengths\_count, long \*lengths, long minLength) { long t=0, i=1; 32 v for(int i=0;i<=lengths\_count-1;i++){ t+=lengths[i];

```
#include<stdio.h>
30 v char* cutThemAll(int lengths_count, long *lengths, long minLength) {
   long t=0, i=1;
32 v for(int i=0;i<=lengths_count-1;i++){
        t+=lengths[i];
35 ▼ do{
        if(t-lengths[lengths_count-1]<minLength){</pre>
            return "Impossible";
        i++:
    }while(i<lengths_count-i);</pre>
    return "Possible";
```

	Test	Expected	Got	
~	<pre>long lengths[] = {3, 5, 4, 3}; printf("%s", cutThemAll(4, lengths, 9))</pre>	Possible	Possible	~
~	<pre>long lengths[] = {5, 6, 2}; printf("%s", cutThemAll(3, lengths, 12))</pre>	Impossible	Impossible	~

return s;

26 27

33 34

36 ▼

37 38 39

41 42 43

```
Z3 |#IIICIUUE\SCUIU.II/
30 v char* cutThemAll(int lengths_count, long *lengths, long minLength) {
31 long t=0, i=1;
32 v for(int i=0;i<=lengths_count-1;i++){
33
        t+=lengths[i];
34
35 ▼ do{
36 ▼
        if(t-lengths[lengths_count-1]<minLength){</pre>
37
            return "Impossible";
38
39
        i++:
    }while(i<lengths_count-i);</pre>
    return "Possible":
41
42
43
```

	Test	Expected	Got		
~	<pre>long lengths[] = {3, 5, 4, 3}; printf("%s", cutThemAll(4, lengths, 9))</pre>	Possible	Possible	~	
~	<pre>long lengths[] = {5, 6, 2}; printf("%s", cutThemAll(3, lengths, 12))</pre>	Impossible	Impossible	~	