

Week 15

The image shows two screenshots from a web browser. The top screenshot is a Moodle 'My courses' page for a user named MUKILRAJ S 2024-CSE. The page title is 'My courses' and the breadcrumb is 'Dashboard / Site pages / My courses'. On the left is a navigation menu with 'Dashboard', 'Site pages' (containing 'My courses', 'Site blogs', 'Site badges', 'Tags', 'Site announcements'), and 'My courses' (containing 'GE23131-PUC-2024'). The main content area is titled 'Course overview' and shows a course card for 'GE23131-Programming Using C-2024' in 'Category 1', which is '51% complete'. The bottom screenshot is a quiz attempt review for 'Week-15-Pointers: Attempt review'. It shows 'Question 1' as 'Correct' and 'Marked out of 1.00'. The question text is: 'Given an array of integers, reverse the given array in place using an index and loop rather than a built-in function.' It includes an example with an array [1, 3, 2, 4, 5] and its reverse [5, 4, 2, 3, 1]. The 'Function Description' asks to complete a function `reverseArray` that takes an array of integers and returns the array in reverse order. 'Constraints' are $1 \leq n \leq 100$ and $0 < arr[i] \leq 100$. 'Input Format For Custom Testing' specifies the first line is n and subsequent lines are the array elements. 'Sample Case 0' shows sample input: 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

Week-15-Pointers: Attempt review | REC-CIS - Personal - Microsoft Edge

Not secure | www.rajalakshmicolleges.org/moodle/mod/quiz/review.php?attempt=159966&cmid=404

REC-CIS

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

29°C Haze

Search

ENG IN

15:24 14-01-2025

```

32  * }
33  *
34  */
35  int* reverseArray(int arr_count, int *arr, int *result_count)
36  {
37  *result_count=arr_count;
38  for(int i=0;i<arr_count/2;i++)
39  {
40  int temp=arr[i];
41  arr[i]=arr[arr_count-i-1];
42  arr[arr_count-i-1]=temp;
43  }
44  return arr;
45  }
46

```

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

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

- $2 \leq n \leq 10^5$
- $1 \leq t \leq 10^9$
- $1 \leq \text{lengths}[ij] \leq 10^9$
- 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 number of elements in *lengths*.

Each line i of the n subsequent lines (where $0 \leq i < n$) 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.

```
1 1 /*
2 2 * Complete the 'cutThemAll' function below.
3 3 *
4 4 * The function is expected to return a STRING.
5 5 * The function accepts following parameters:
6 6 * 1. LONG_INTEGER_ARRAY lengths
7 7 * 2. LONG_INTEGER minLength
8 8 */
9 9
10 10 /*
11 11 * To return the string from the function, you should either do static allocation or dynamic allocation
12 12 *
13 13 * For example,
14 14 * char* return_string_using_static_allocation() {
15 15 *     static char s[] = "static allocation of string";
16 16 *     return s;
17 17 * }
18 18 *
19 19 * char* return_string_using_dynamic_allocation() {
20 20 *     char* s = malloc(100 * sizeof(char));
21 21 *     s = "dynamic allocation of string";
22 22 *     return s;
23 23 * }
24 24 *
25 25 *
26 26 * }
27 27 */
28 28 char* cutThemAll(int lengths_count, long *lengths, long minLength)
29 29 {
30 30     long t=0,i=1;
31 31     for(int i=0;i<=lengths_count-1;i++)
32 32     {
33 33         t+=lengths[i];
34 34     }
35 35     do{
36 36         if(t >= minLength){
37 37             return "YES";
38 38         }
39 39         t=0;
40 40         i=1;
41 41         for(int i=0;i<=lengths_count-1;i++)
42 42         {
43 43             t+=lengths[i];
44 44         }
45 45     }while(t <= minLength);
46 46     return "NO";
47 47 }
```

```

30 {
31     long t=0,i=1;
32     for(int i=0;i<=lengths_count-1;i++)
33     {
34         t+=lengths[i];
35     }
36     do{
37         if(t-lengths[lengths_count-i-1]<minLength)
38         {
39             return "Impossible";
40         }
41         i++;
42     }
43     while(i<lengths_count-1);
44     return "Possible";
45 }
46
47 }
48

```

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

Passed all tests! ✓