## REC-CIS

GE23131-Programming Using C-2024

|  |  |
| --- | --- |
| **Status** | Finished |
| **Started** | Monday, 13 January 2025, 6:57 PM |
| **Completed** | Monday, 13 January 2025, 7:10 PM |
| **Duration** | 12 mins 16 secs |

Given an array of integers, reverse the given array in place using an index and loop rather than a built-in function.



Question **1**

Correct

Marked out of 1.00

Flag question

Quiz navigation

1

2

[Show one page at a time](http://www.rajalakshmicolleges.org/moodle/mod/quiz/review.php?attempt=157765&cmid=404&showall=0)

Finish review

# 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 ≤ n ≤ 100*

*0 < arr[i] ≤ 100*

# Input Format For Custom Testing

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

REC-CIS

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

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

## REC-CIS

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

1 ▼

2

3

4

5

6

7

8 ▼

9

10

11

12

13

14 ▼

15

16

17

18

19

20

21

22 ▼

23

24

25

26

27 ▼

28

29

30

31

/\*

* Complete the 'reverseArray' function below.

\*

* The function is expected to return an INTEGER\_ARRAY.
* The function accepts INTEGER\_ARRAY arr as parameter.

\*/

/\*

* To return the integer array from the function, you should:
* - Store the size of the array to be returned in the resu

l

t

l

* - Allocate the array statically or dynamically
* For example,
* int\* return\_integer\_array\_using\_static\_allocation(int\* resul
* \*result\_count = 5;

\* static int a[5] = {1, 2, 3, 4, 5};

* return a;
* }
* int\* return\_integer\_array\_using\_dynamic\_allocation(int\* resu
* \*result\_count = 5;
* int \*a = malloc(5 \* sizeof(int));
* for (int i = 0; i < 5; i++) {

\* \*(a + i) = i + 1;

* }
* return a;

## REC-CIS

35

36

37 ▼

38

39

40 ▼

41

42

43

44 ▼

#include<stdio.h> #include<stdlib.h>

int\* reverseArray(int arr\_count, int \*arr, int \*result\_count) { int \*result = (int\*)malloc(arr\_count \* sizeof(int));

if(result ==NULL){ return NULL;

}

for(int i=0;i<arr\_count;i++)

{

nt-i-1];

32 \* }

33 \*

34 \*/



Question **2**

Correct



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

|  |  |  |
| --- | --- | --- |
| 45 |  | result[i]=arr[arr\_cou |
| 46 |  | } |
| 47 |  | \*result\_count=arr\_count; |
| 48 |  | return result; |
| 49 |  |  |
| 50 |  |  |
| 51 | } |  |
| 52 |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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 |  |

## REC-CIS

Marked out of if it is possible to make the necessary cuts using this machine. The rod is marked into

1.00 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.

Flag question

## REC-CIS

*cutThemAll* has the following parameter(s):

**Sample Case 0**

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

machine.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer, *lengths[i]*.

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

**Input Format For Custom Testing**

*2 ≤ n ≤ 105*

*1 ≤ t ≤ 109*

*1 ≤ lengths[i] ≤ 109*

*The sum of the elements of lengths equals the uncut rod length.*

·

·

·

·

Constraints

Returns

string: *"Possible"* if all *n-1* cuts can be made. Otherwise, return the string *"Impossible"*.

*int lengths[n]*: the lengths of the segments, in order

*int minLength*: the minimum length the machine can accept

REC-CIS

**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

## REC-CIS

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.

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

1 ▼ /\*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 |  |  | \* | Complete the 'cutThemAll' function below. |
| 3 |  |  | \* |  |
| 4 |  |  | \* | The function is expected to return a STRING. |
| 5 |  |  | \* | The function accepts following parameters: |
| 6 |  |  | \* | 1. LONG\_INTEGER\_ARRAY lengths |
| 7 |  |  | \* | 2. LONG\_INTEGER minLength |
| 8 |  |  | \*/ |  |
| 9 |  |  |  |  |
| 10 | ▼ |  | /\* |  |
| 11 |  |  | \* | To return the string from the function, you should either do st |
| 12 |  |  | \* |  |
| 13 |  |  | \* | For example, |
| 14 | ▼ |  | \* | char\* return\_string\_using\_static\_allocation() { |

1. \* static char s[] = "static allocation of string";
2. \*

17

\* return s;

## REC-CIS

18 \* }

19 \*

20 ▼ \* char\* return\_string\_using\_dynamic\_allocation() {

21

22

23

24

25

26

27

28

29

30

31 ▼

32

33 ▼

34

35

36 ▼

37 ▼

38

39

40

41

42

43

44

45

* char\* s = malloc(100 \* sizeof(char));

\*

* s = "dynamic allocation of string";
* return s;
* }

\*/

#include<stdio.h>

char\* cutThemAll(int lengths\_count, long \*lengths, long minLength) long t=0,i=1;

for(int i=0;i<=lengths\_count-1;i++){ t+=lengths[i];

}

do{

if(t-lengths[lengths\_count-1]<minLength){ return"Impossible";

}

i++;

}

while(i<lengths\_count-i); return "Possible";

}



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **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 |  |

## REC-CIS

Passed all tests! 

Finish review