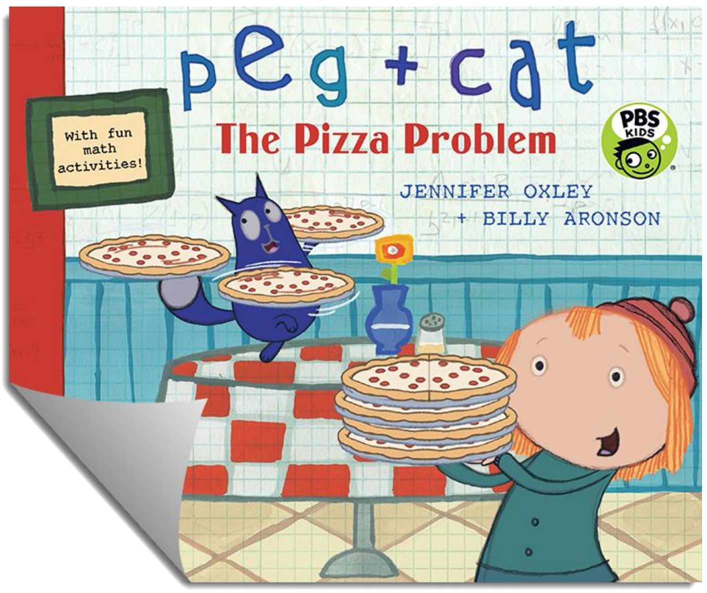
Practice Problems for PE08

For PE08, you'll solve one problem which will ask you to write a function to process partially-filled arrays. The problem will involve insertion and/or deletion into a partially-filled array. Here are some practice programming problems. Remember that the actual problem you get on the test will be similar to one of these, but not necessarily the same.

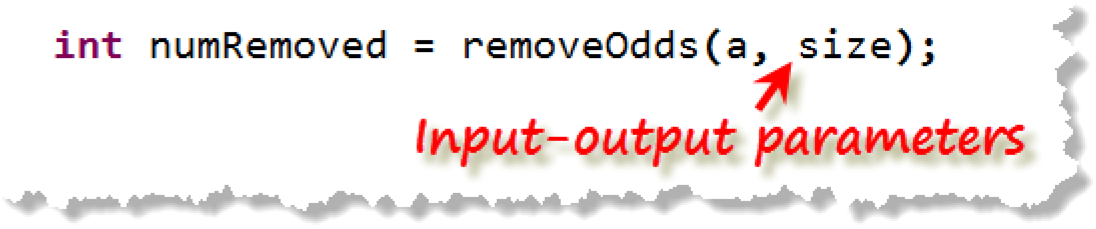


1 THE removeOdds PROBLEM

Write the function `removeOdds()` that removes all of the removes all of the odd numbers in its array parameter. The function returns the number of items removed, or -1 if the array is initially empty. The function takes 2 arguments:

- the array of `int` that will be modified
- the size of the array (use `size_t` as the type)

Both of these will be *input-output* parameters.



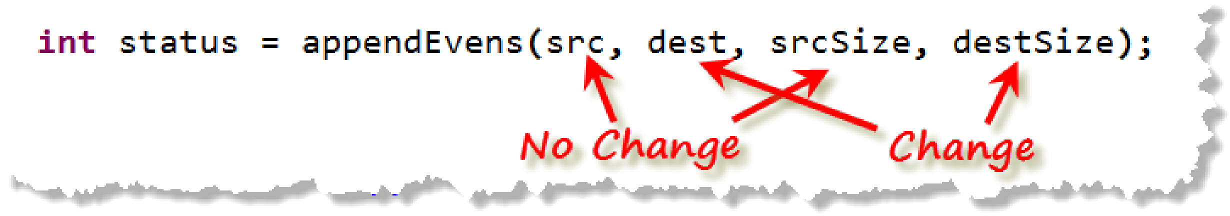
```
int numRemoved = removeOdds(a, size);
```

2 THE appendEvens PROBLEM

Write the function appendEvens() that, given two arrays, add all of the even elements from the first into the second. You may assume the second has enough space. Here's an example:

```
a1[1, 2, 3], a2[7, 9]-> a1[1, 2, 3], a2[7, 9, 2]
```

The function returns the number of items appended, or **-1** if the first array is empty.



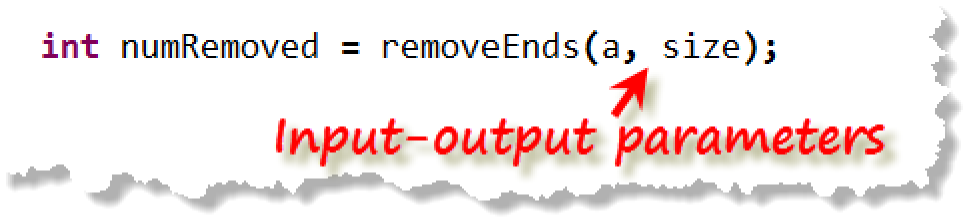
```
int status = appendEvens(src, dest, srcSize, destSize);
```

3 THE removeEnds PROBLEM

Write the function removeEnds() that removes the first and last element in the array, except that, if the array size is odd, it removes the first two elements along with the last element. The function takes 2 arguments:

- the array of int that will be modified
- the size of the array (use size_t as the type)

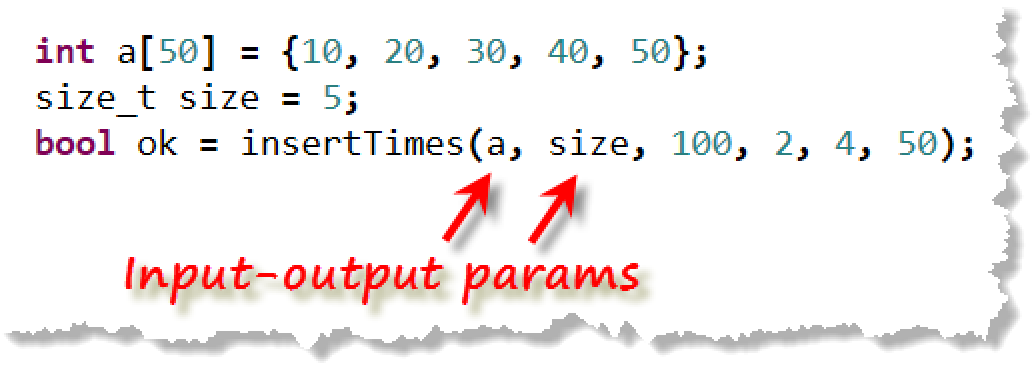
Both of these may be changed by the function. The function returns the number of items removed or **-1** if the array has too few elements.



```
int numRemoved = removeEnds(a, size);
```

4 THE insertTimes PROBLEM

Write the function insertTimes() which inserts a given number at a given index for a given number of times. For instance, given the array and function call below, which inserts the value 100 into the array a, repeating it 4 times. The capacity of the array is 50.



```
int a[50] = {10, 20, 30, 40, 50};
size_t size = 5;
bool ok = insertTimes(a, size, 100, 2, 4, 50);
```

As you can see, the function takes 6 arguments:

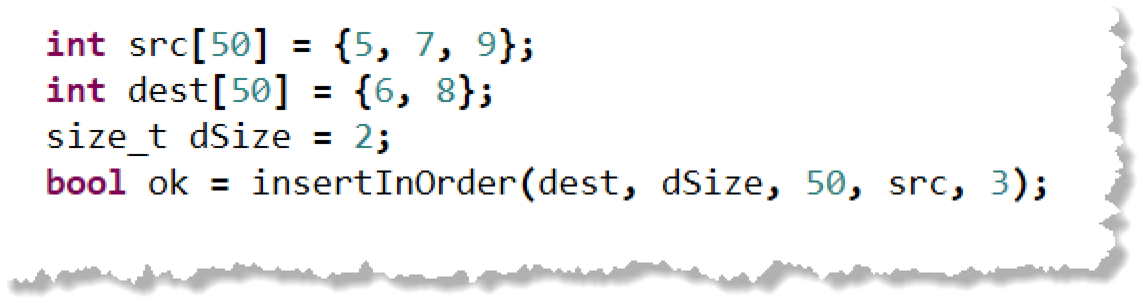
- the array of `int` and the size of the array. Both of these *may be modified*.
- the `value` to insert (100), the `position` to insert at (2), the `number` of times to insert the value (4), and the capacity of the array (50).

All of these should be type `size_t` except for the `value` which is inserted into the array (and the array elements themselves), which should be of type `int`.

The function returns `true` if the elements can be inserted and `false` if they cannot. The insertion could fail because the insert position appeared past the size (although it should be possible to insert immediately after the last element), or if adding the elements would exceed the capacity of the array.

5 The `insertInOrder` Problem

Write the function `insertInOrder()` which inserts each of the elements in the `source` array into the ordered position in the `destination` array. You may **assume** that the `destination` array is already in sorted order. Here is a short example:



```
int src[50] = {5, 7, 9};
int dest[50] = {6, 8};
size_t dSize = 2;
bool ok = insertInOrder(dest, dSize, 50, src, 3);
```

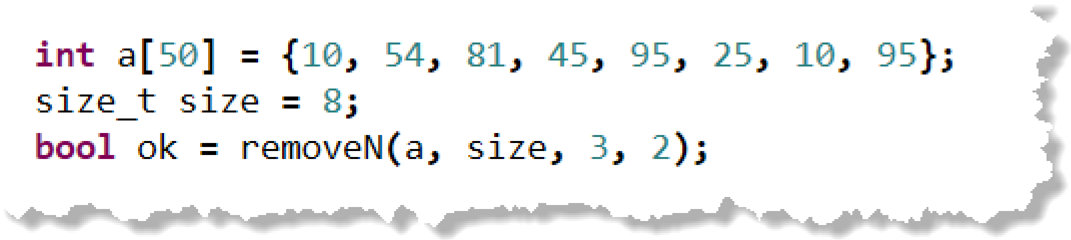
As you can see the function takes 5 arguments:

- The destination array and its size, which may both be modified.
- The destination capacity, the source array and the source size, which are not modified.

In the example shown above, the function will return `true` (since it can succeed) and the resulting array will contain: `{5, 6, 7, 8, 9}`. The new `dSize` will be `5`. The function returns `false` if it fails. The only reason it could fail is if inserting the new elements would cause the capacity to be exceeded, in which case the destination array should not be changed in any way.

6 THE `removeN` PROBLEM

Write the function `removeN()` which removes "N" elements from the array starting at a given position. Here is an example where we remove 3 elements, starting at position 2.



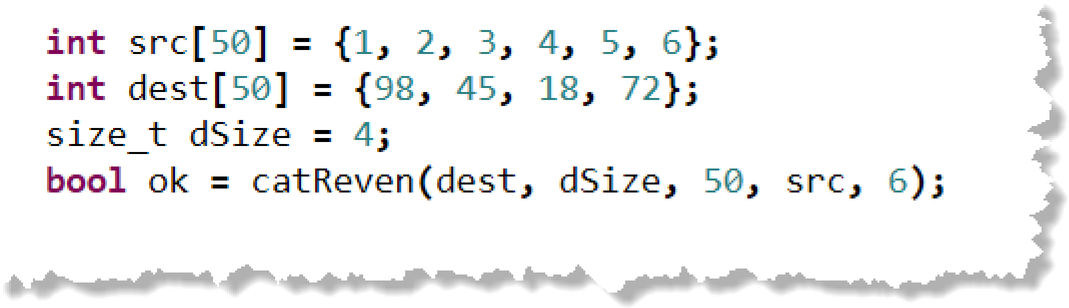
```
int a[50] = {10, 54, 81, 45, 95, 25, 10, 95};
size_t size = 8;
bool ok = removeN(a, size, 3, 2);
```

The function takes four parameters: the array and its effective size, which may both be modified, along with the number of elements to remove and the position to start removing. (Be careful with these, that you don't reverse them.)

The function returns true if it succeeds and false otherwise. It can fail if there are not N elements remaining (measured from the starting position), or, if position is out of bounds.

7 THE `catReven` PROBLEM

Write the function `catReven()` which concatenates (appends) all the even numbers of the source array to the destination array, in reversed order. Here's a short example:



```
int src[50] = {1, 2, 3, 4, 5, 6};
int dest[50] = {98, 45, 18, 72};
size_t dSize = 4;
bool ok = catReven(dest, dSize, 50, src, 6);
```

As you can see the function takes 5 arguments:

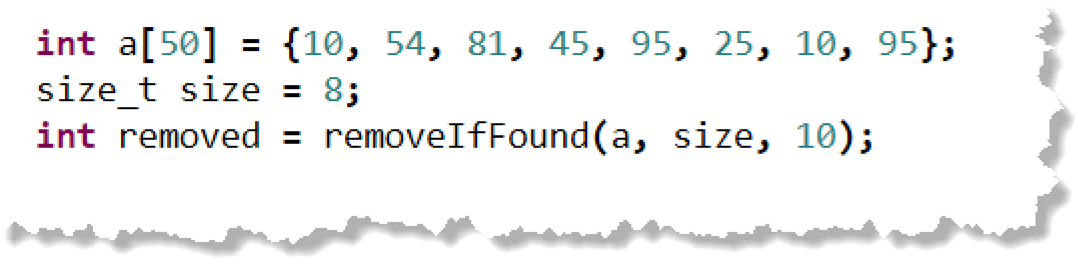
- The destination array and its size, which may both be modified.

- The destination capacity, the source array and the source size, which are not modified.

In the example shown `dSize` would be changed to 7 and the destination array would contain `{98, 45, 18, 72, 6, 4, 2}`. The function returns `true` if successful and `false` if the destination array does not have the space to add the new elements, or if the source array is empty.

8 THE `removeIfFound` PROBLEM

Write the function `removeIfFound()` which removes all copies of a particular value from an array. The function returns the number of values removed. Here's a short example:



```
int a[50] = {10, 54, 81, 45, 95, 25, 10, 95};
size_t size = 8;
int removed = removeIfFound(a, size, 10);
```

The example shown here removes all copies of the value `10` from the array a. Both a and `size` are modified by the function. In this case, since there are two copies of `10`, `size` is changed to 6 and the function returns 2. Be especially careful when writing your code that you don't skip values that are contiguous in the array. For instance, if you remove 2 from `{1, 2, 2, 2, 1}`, you should end up with `{1, 1}`.