Arrays & Algorithms

There are three short functions for this assignment.

1. Alternating Sum

An alternating sum is one where you add the first element, subtract the second, add the third, and so on. It is a type of cumulative algorithm, which you met in this chapter.

Here's an example:

> *If* `alternatingSum()` *is passed an array:* `1 4 9 16 9 7 4 9 11`
> *it returns:* `1 - 4 + 9 - 16 + 9 - 7 + 4 - 9 + 11  (or -2)`

Check your assignment using `make test`. If you get stuck, ask for help on the Discussion Board.

2. Extreme Values

The second function is `minMax()`. Here is the specification:

> *The* `minMax` *function takes a pointer to an element in an array of* `double` *and the number of subsequent elements to process. It returns a* `MinMax` *variable containing a* ***pointer to the maximum*** *value in the sequence and* ***a pointer to the minimum*** *value in the sequence.* ***You are not*** *to return the largest and smallest values themselves. If* `size` *is* `0`*, the special pointer value* `nullptr` *(or* `0`*) you be used for both min and max.*

Read about the **extreme values** algorithm in this chapter. As mentioned, you **don't** want to return the largest (or smallest) value; you want to return **the addresses of the largest and smallest value**.

Here's a modified version of the algorithm:

```
Pointer ptr points to first element
Structure result {min=ptr, max=ptr}
Pointer atEnd <- address past the end of the sequence
Increment ptr // point to second element
While the ptr < atEnd Do
   If *ptr > *result.max Then result.max <- ptr
   If *ptr < *result.min Then result.min <- ptr
   Increment ptr
Return result (smallest & largest inside structure)
```

I've used indenting to reflect the general structure of the function.

Writing the Code

I have already stubbed out the function. Below, I have notes on the code for each of those sections.

1. Assign the address of the first element to `ptr`
2. Assign `ptr` to `result.min` and `result.max`.
3. Create the `atEnd` pointer. Add `size` to `ptr`.
4. Increment `ptr` to point to the second element
5. **Traversing the array.** In a `while` loop, if `ptr` is < `atEnd` Do
   a. **Comparing the values**. Inside the loop body, dereference `result.min` and `result.max` and compare those values to the value that `ptr` is pointing at. Update the structure members depending on the results.
   b. Move the pointer to the next element using the **increment operator**.

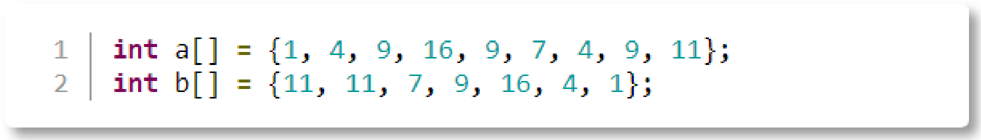
I'm sure you can do the final step on your own. If you run the tests now, you'll see you have **90%**. The test that fails is when `size` is `0`.

For that, you need to simply add a special case before the loop begins.

When you fix that and `make test`, all of the tests should pass.

3. The Same Set

Write a predicate function `sameSet()` that checks whether two arrays have the same elements in some order, ignoring multiplicities. For example:



```
int a[] = {1, 4, 9, 16, 9, 7, 4, 9, 11};
int b[] = {11, 11, 7, 9, 16, 4, 1};
```

Two arrays.

The two arrays here have the same set, because both have the same elements, even though the first one has three nines, and the second has two elevens.

The function should take two pairs of **begin-end iterators** to the beginning and one-past-the-end of the two arrays. Remember that a value may be found in the first array, and not be in the second, or, a value may be found in the second array and not appear in the first. In both cases your function will return `false`.

You may create one or more helper functions if you like. You can get help on the discussion board **if you get stuck**.

Be sure to `make submit` to turn in your code for credit **before the deadline**. As always, if you run into problems, bring your questions to the Discussion Board, or come to my office hour.