Homework

24

More Arrays

For this assignment you'll to write two functions that process arrays. You'll use iterator loops as well as partially-filled arrays. Here are the functions:

1. The **`sameSet()`** function uses **begin-end iterators** to tell if two array segments have the same set of values (ignoring order and multiplicity).
2. The **`copyEvens()`** function that copies all of the even elements in the partially-filled array **`a`** into the partially-filled array **`b`**.

1. The Same Set

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

```
1  int a[] = {1, 4, 9, 16, 9, 7, 4, 9, 11};
2  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 Piazza if you get stuck.

2. Copy Evens

Here is the specification for the function **copyEvens()** which copies all of the even-valued elements in the array **a** into the array **b**.

```
1  void copyEvens(const int a[], size_t aSize,
2                 int b[], size_t& bSize);
```

The copyEvens prototype.

- On entry, **bSize** will contain the declared size of **b**.
- Make sure that **bSize >= aSize**, and throw the standard **length_error** if it is not, along with an appropriate error message.
- Note that **b** is not constant, (because it may be changed). Your function should set **bSize** to the actual number of elements copied.

Stub out the function. Test it and you should get a few points. Since the function doesn't return anything, all you have to do is remove the semicolon and add braces.

Step 1 – Check the Parameters

To check the parameters, you just need to make sure that **bSize** is at least as long as **aSize**. If it is not, then **we don't have enough room to store all the values** inside **a**, in the event that **a** is composed entirely of even numbers. If you fail this test, then you are going to **throw an exception**. (Notice that the starter code has already included **<stdexcept>**. If it had not done so, you would have to remember it.)

The standard exception that you want to throw is called **length_error**. Display a simple message in this case.

Step 2 – Visit Every Element in Array a

What **kind of loop** should you use to visit every element in the array **a**? You have your choice: use **either** an array-notation, counter-controlled loop, or, even a pointer or iterator-style range-based loop.

```
1  for (size_t i = 0; i < aSize; i++)...
2  for (auto p = a, end = a + aSize; p != end; p++)...
```

Possible loop bounds.

For the range-based loop, you have to "manufacture" the **end** value yourself; you can't call the **end()** function like you can when the array is in scope; remember that the

parameter variable **a** is actually a pointer, **not an array**, so that the function **end()** does not work on it.

Step 3 – The Index into Array b

Now you have a loop that visits every element in **a**, but you don't have an index into **b** so we can store the odd numbers as they are found. The easiest way to handle this is to use the parameter reference variable **bSize**. This is an **input-output variable**:

- Check its **input value** that **b** is large enough to hold all the values in **a**.
- Its **output value** is the number of items copied from **a** to **b**.

Since at this point you have copied **0** elements, set it to **0** and then use it as the index into **b**. As an alternative, you may create another pointer pointing to the first element of **b**, and then use **pointer difference** to set the finished **bSize** value.

Here are both ways:

```
1  bSize = 0;
2  for (size_t i = 0; i < aSize; i++)...
3
4  auto pb = b;
5  for (auto p = a, end = a + aSize; p != end; p++)...
```

Accessing array b.

Step 4 – Extract the Odd Numbers

Again, you have **two ways to extract the numbers**; for the array-notation version use the **subscript operator**. For the pointer-notation code, use **dereferencing**. Here's the code for both methods.

```
1  if (a[i] % 2 == 0) b[bSize++] = a[i];
2  if (*p % 2 == 0) *pb++ = *p;
```

Extracting the odd numbers.

Step 5 – The Length of Array b

The length of array **b** is correct when you leave the array-notation version of the loop, since you are using the output variable **bSize** as the index into the array **b**. For the pointer-notation version, though, you'll need to **add one more statement**. After the loop, calculate the correct output value for **bSize** using pointer difference, like this:

```
bSize = pb - b;
```

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

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 Piazza or come to my office hour.