Homework

26

More Arrays

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

1. 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.

2. cliqueCount

Write a function that counts the number of **"cliques"** (two or more adjacent elements with the same value) that an array contains. Here are three examples

```
cliqueCount({1, 2, 2, 3, 4, 4}, 6) → 2
cliqueCount({1, 1, 2, 1, 1}, 5) → 2
cliqueCount({1, 1, 1, 1, 1}, 5) → 1
```

The first example has the cliques **2,2** and **4,4**. The second example has two cliques, both of which contain **1,1**. While the final example is a single clique of **1,1,1,1,1**. You may create one or more helper functions if you like.

3. sevenEleven

You are given an array that contains exactly the same number of **7**s and **11**s. Rearrange the elements so that every **7** is immediately followed by an **11**. You may not move the **7**s, but every other number may move. When the function is called, every **7** is followed by a number which is not an **11**, and a **7** appears in the array before any **11**.

```
sevenEleven({1, 7, 1, 11}) → {1, 7, 11, 1}
sevenEleven({1, 7, 1, 11, 11, 7, 1}) → {1, 7, 11, 1, 1, 7, 11}
sevenEleven({7, 2, 2, 11}) → {7, 11, 2, 2}
```

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.