CS 170 | Spring 2017
Programming Assignment #6

Files (submit folder) due
- Monday, June 26, 2017
- 01:00am

Checklist due
- At the start of class

Remember: The major point of this assignment is for you to learn how to submit your homework correctly. You should strive to follow all of the directions exactly as specified in the handouts. The syllabus that was handed out during the first day of class also contains important information on how to submit your homework. If you are still unsure of how to do something, you should ask for help, either from myself or from a tutor in the lab or from another student.

Information

This assignment gives you practice with function templates, pointers and ranges. The goal is to implement several templated functions that work on ranges. You will implement functions that perform operations on ranges such as removing elements within the range, replacing elements within the range, searching for elements, copying one range to another range, etc. Many of these functions mimic how the generic algorithms in the STL work. You will have a header file that looks like this:

```
//-----------------------------------------------------------------------------
#ifndef FUNCTIONS_H
#define FUNCTIONS_H
//-----------------------------------------------------------------------------
namespace CS170
{
 /*
 * Other template function declarations for count, remove, replace, etc.
 * go here.
 *
 */
template <typename T> void swap(T &left, T &right);
#include "Functions.cpp"
}
#endif
//-----------------------------------------------------------------------------
```

Your .cpp file will contain several functions. The sample driver shows many of them. You will need to figure out what others are required. Many of the functions are going to be very similar in their implementation. However, for this assignment, don't be tempted to try and factor out the minimal common code into a separate function, because it will only complicate matters. Once you understand the concept of a range, you will see that the amount of code is not that great. (It never is.) The most complex function is the remove function, so you should work on that one last.

As you implement the functions, you should begin to see a pattern emerging in your code. This should help you understand the purpose of using pointers (a range) with these arrays instead of relying on the size. A range is much more flexible than an array with a size, specifically because it allows you to work on a part of the array (a range) rather than the entire array.

Other criteria

1. With the exception of the remove function, all of the functions are trivial, requiring about 4 or 5 lines of code.
2. Do not use the subscript operator anywhere in your code. (Not even in a comment.) You are given a range, not an array.
3. You are not to use for loops anywhere in your code. Use while loops. (Many students still don't understand the while loop.)
4. You must make sure that your functions can deal with the appropriate calls. This means you need to decide how and when to use const in your code. The sample driver may not test all cases so you may need to add more tests. You have been warned.
5. Remember: templated functions are only generated if you call the function. Don't end up getting a 0 due to your code failing to compile because you didn't add the necessary test cases. Your code will fail to compile if you forgot a case.

6. Templated functions can take many different types, even large user-defined types. Use references wherever possible.
7. Do not include any header files other than iostream in your .cpp file. (You don't need any others.)
8. Notice that you are not creating any classes for this assignment, just a bunch of functions. In your files, make sure that you arrange the functions alphabetically. (This includes the header file as well as the implementation file. This will make it easier to find the functions when we grade your assignment.)
9. Your template parameter must be named T. If you have two template parameters, they must be named T1, and T2.
10. You will need to use std::cout to print out the elements in the display function. Make sure you format the output exactly as shown in the Output.txt file as I am not providing the output function for this assignment. Not using a diff tool can potentially cause all of your output to be incorrect. Again, you have been warned.
11. You must include the .cpp file at the end of the .h file exactly as shown above.
12. The remove function is the only non-trivial function and should be done last. You must write pseudocode. If you can't write the pseudocode (meaning that you don't understand what you are trying to do), then you can't write C++ code. Make sure that your function only makes one pass over the array. Failure to do so will cause you to lose points.
13. Finally, NO OVERLOADED FUNCTIONS ARE ALLOWED. This means that some functions will need multiple template parameters to work correctly.

Range Notes

When we are talking about a range of objects in a container (a container can be any kind of container, e.g. array, linked list, etc.), we are actually describing a half-open range. This means that the left end "points" to the first element in the range and the right end "points" to one after the last element to include. (left and right may not be pointers, but you can think of them that way.) This is also called a left-inclusive range and is noted like this:

[first, last)
So, if you wanted to print the middle 3 integers of this array, this is how you might do it:

```
int array[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
print_range(&a[3], &a[6]); // some function that will print a range of ints
```

You are given a partial .h, .cpp and Driver to start with as well as some Sample output. You should make sure to do extra test cases for your code to make sure it can handle many types and levels of const.

Popular Questions and Answers for this assignment.

Question:

In the copy function, you provide two ranges, but only the first one has an end. What happens if the second range isn't large enough to hold all of the items? Do we have to check for that case?

Answer:
You are not given an end to the second range because you don't need it. If the first range is larger than the second range, *bad things will happen*. This is by design. Remember, this is C++ and the programmer is always right (and speed comes first.) Also, realize that you **cannot** check to see if the second range is large enough as you only are given a pointer to the start.
Another reason why we aren't going to worry about bad behavior is because this is how the STL is implemented, and you are trying to understand How Things Work™.

Question:
Are we allowed to create a new array for any of the functions?
Answer:
Absolutely not. Not now, not ever. No. All of the modifications are to be made on the ranges that you are provided. You are not to create any arrays. **In fact, you are not to use the subscript operator [ ] in any code**. Why? Because you may not be given an array. Remember, this is a template function and you have no idea what data structure will be provided.

Question:
How many functions are we supposed to implement? Some of my functions can be overloaded or can be written with multiple template parameters.
Answer:
The driver shows you what functions need to be created. (This is sort of what's called *Test Driven Development*, where you're given the test cases and you need to make them all work.) As stated in the handout, NO OVERLOADED FUNCTIONS ARE ALLOWED. As you said, if you are using only one template parameter, **`T`**, then some functions may need to be overloaded, but you're not allowed to do that. These functions can use two template parameters, **`T1`** and **`T2`**, and won't need to be overloaded.

- Sample command lines for compiling: (notice that `Functions.cpp` is not mentioned)

  GNU:

  ```
  g++ -o gnu.exe driver-sample.cpp -O -Wall -Wextra -ansi -pedantic
  ```

  Microsoft

  ```
  cl /W4 /EHa /WX /Za /MT /O2 /Fems.exe driver-sample.cpp
  ```

  Borland (notice the **`-w-8092`** to remove bogus 'iterator' warnings)

  ```
  bcc32 -w -v -vG -w-8092 -ebor.exe driver-sample.cpp
  ```

- **If you've implemented your functions properly, you should be able to finish these tests in less than 2 seconds.**

- Here's an example: (What are the types of `begin`, `end`, and `value`?)

```
template<typename T>
int count(const T *begin, const T *end, const T& value)
{
  int counter = 0;
  while (begin != end)
  {
    if (*begin == value)
      counter++;
    begin++;
  }
  return counter;
}
```

What operations must `T` support for the code above to compile?

More details on the **`remove`** function:

```
void TestRemove(void)
{
    // Initialize an array with 10 elements
  int i1[] = {1, 50, 50, 4, 5, 50, 7, 8, 50, 10};

    // Print all of them
  CS170::display(i1, i1 + 10);

    // Remove all occurrences of 50, returns a
    // pointer to the new 'end'
  int *newend = CS170::remove(i1, i1 + 10,  50);

    // Print out the list (only up to newend)
  cout << "remove 50, new list: ";
  CS170::display(i1, newend);

    // Print all items out
  CS170::display(i1, i1 + 10);
}
```

Output:

```
1, 50, 50, 4, 5, 50, 7, 8, 50, 10
remove 50, new list: 1, 4, 5, 7, 8, 10
1, 4, 5, 7, 8, 10, 7, 8, 50, 10
```

A closer look at the modified array:

```
  1   4   5   7   8   10   7   8   50   10
  ^                  ^             ^
begin              new end      real end
```

Usually, when we remove elements from an array, we must shift all of the elements to the right of the deleted item over to the left. Shifting elements (especially in a very large array) is a very expensive operation. The technique that **`remove`** implements does not shift any elements. Instead, it just copies a few elements.

It's also important to notice that the **`remove`** function returns *the new end* (or logical end). This is important because it tells the caller where the end of the remaining elements is. Without that return value, the caller would have no way to know where the "good" elements stop.

Deliverables

You must submit the header file (Functions.h), the implementation file (Functions.cpp).
These file must be zipped up and submitted to the appropriate submission page.

Source Files Description:

Functions.cpp - The implementation file. All implementation for the functions goes here. You must document the file (file header comments) and all functions (function header comments) using proper Doxygen-style comments. Failure to do so will result in a very poor grade. Also, be sure your functions are sorted alphabetically.
Functions.h - The header file. You will need to modify the one I posted. No implementation is permitted in this file. However, you must #include the .cpp file exactly as instructed. Also, be sure your functions are sorted alphabetically.

Usual stuff

Your code must compile (using the compilers specified) to receive credit. The code must be formatted as per the documentation on the website. Make sure your name and other info is on all documents

- What operations must `T` support for the code above to compile?
- The English alphabet:

  a b c d e f g h i j k l m n o p q r s t u v w x y z

  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z