Assignment 2

Due: October 2, 2018 (11:59pm)

Assignment

You will implement and test the sequence class using an array to store the sequence's items.

Purpose

- Ensure that you can write a small class that uses an array as a private member variable.
- Familiarize yourself with the sequence container class (which may also be part of future assignments).
- Give us a chance to evaluate your programming skills on a small class.

Before starting

Read all of Chapter 3.

How to Turn In

Submit all source code files (*.cpp, *.cxx, *.h) in a collection as a zip on Blackboard.

Files that you must write and turn in

- sequence1.h: The header file for the sequence class. You don't have to write much of this file. Just start with the provided version. Decide on appropriate private member variables, and declare these in the sequence class definition at the bottom of the header file. If some of your member functions are implemented as inline functions, then you may put those implementations in this file too.
- sequence1.cpp: The implementation file for this first sequence class. You will write all of this file, which will have the implementations of all the sequence's member functions.

Other files that you may find helpful

- sequence_test.cpp: A simple interactive test program.
- sequence_exam.cpp: A non-interactive test program that will be used to test the correctness of your sequence class.

Further discussion of the assignment

Many of the features of this class are similar to the bag class from Section 3.1, so start by thoroughly reading Section 3.1 and pay attention to new features such as static constant members and the use of a typedef. The sequence class itself is discussed in Section 3.2 of the book. Notice how the sequence differs from a bag. The interactive test program sequence_exam.cpp is discussed in Section 3.3 of the book.

Start by declaring the sequence's private member variables in sequence.h. Then write the invariant of your ADT at the top of sequence1.cpp. The invariant describes precisely how all of your private member variables are used. All of the member functions (except for the constructor) may count on the invariant being true when the member function is activated. And all of the member functions are responsible for ensuring that the invariant is true when the function returns.

Do your work in small pieces. Start with the bare minimum to get the code to compile and add on pieces, testing that each new complete piece compiles.

Use the interactive test program and a debugger to track down errors in your implementation. If you have an error, do not start making changes until you have identified the cause of the error.