Assignment 3

Due: October 16, 2018 (11:59pm)

Assignment

You will implement and test a revised sequence class that uses a dynamic array to store the items.

Purpose

Ensure that you can write a small class that uses a dynamic array as a private member variable.

Before starting

Read all of Chapter 4, with special attention to Chapter 4.3.

How to Turn In

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

Files

- `sequence2.h`: The header file for the new sequence class. You don't have to write much of this file. Just start with the provided version. If some of your member functions are implemented as inline functions, then you may put those implementations in this file too. You might want to compare this header file with your first sequence header file (sequence1.h), and likely use that previous work as a starting point for this code. The new version no longer has a `CAPACITY` constant because the items are stored in a dynamic array that grows as needed. But there is a `DEFAULT_CAPACITY` constant, which provides the *initial* size of the array for a sequence created by the default constructor.

- `sequence2.cpp`: The implementation file for the new sequence class. You will write all of this file, which will have the implementations of all the sequence's member functions.

- `sequence2_test.cpp`: This is the same interactive test program that you used with the earlier sequence.
- `sequence2_exam.cpp`: A non-interactive test program that will be used to grade the correctness of your new sequence class.
- `main.cpp`: A simple main file you can use during your development to test your incomplete code.

Description

Your sequence class for this assignment will differ from the your previous sequence in the following ways:

- The number of items which may be stored in the sequence should only be limited by the amount of memory available on the heap. When new items are added to a sequence which is at capacity, the size of the data array in which items are stored should be automatically enlarged.
- Because you are dynamically allocating memory within your sequence class, you will need to define a copy constructor, an assignment operator, and a destructor.
- The constructor should have a default argument which allows the user to set the initial capacity of the sequence.
- There should be a resize function that allows the user to explicitly set the the capacity of the sequence.

Start by declaring the new sequence's private member variables in `sequence2.h`. This should include the dynamic array (which is declared as a pointer to a `value_type`). You will also need two `size_type` variables to keep track of the number of items in the sequence and the total size of the dynamic array. One final `size_type` will allow you to keep track of which item is the current item. After you've declared your member variables, write an invariant for the top of `sequence2.cpp`.

Many of the features of this class are similar to the mintinlinec++bag class from Section 4.3, so start by thoroughly reading Section 4.3 and pay attention to new features such as how the sequence differs from a bag. Also the implementation of some of the functions are almost the same as in the first sequence. Once again, do your work in small pieces. Start by writing mostly empty functions (if a function needs to return a number, start by having it return 0 and do nothing else, etc.) so that you can first get the code to compile, then add to the functions from there.

Use the interactive test program and the 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.

When a member functions needs to increase the size of the dynamic array, it is a good idea to increase that size by at least 10% (rather than by just one item).

Optional (for extra 10% points)

Add the following additional member functions

1. Operators + and +=: For + operator, `x + y` contains all the items of `x`, followed by all the items in `y`. The statement `x += y` appends all the items in `y` to the end of what's already in `x`.

2. Operator `[]`: For a sequence `x`, we would like to be able to refer to the individual items using the usual C++ notation for arrays. For example, if `x` has three items, then we want to be able to write `x[0]`, `x[1]` and `x[2]` to access these three items. The use of the square brackets is called the subscript operator. The subscript operator may be overloaded as a member function, with the prototype shown here as part of the sequence class:
   ```
   value_type operator[](size_type index) const;
   ```
   The only parameter is the index of the item we want to retrieve. The implementation of this member function should check that the index is valid, and then return the specified item.