Homework 2: Deque based on an Orderable Array (the ESA)

This homework requires you to create a Deque class (which collects Student ptrs), which instead of wrapping a Linked List, "wraps" the Student Orderable Array (ESA) you created in Homework #1.

We do not want the DQ to issue ESA "prepend" commands on every **push front** or ESA "remove" commands at index 0 on every **pop front,** because of the huge data shifts that result. We would like the DQ use the ESA in such a way as to avoid these massive data shifts, and never have to worry about array overflow and reallocation / resizing (leaving it to the ESA). If we can do that, the ESA is a better Data Structure choice than a Linked List for the DQ Data Abstraction.

Then you can write the **Deque Iterator (DQI)** class which contains a private pointer to an Orderable Array (ESA), and wraps it. Wrapping means the DQI constructor creates the ESA, and the DQI object implements its public methods by using the public methods of that internal Student ESA.

< Option: You can wrap a standard Array (but do your own reallocs) **>**

<Note: If you used templates for the StudentESA, you should use them again here

```
#include "StudentESA.h"

class StudentDQI  {  // Interface
  private:   // Uncomment one of the following 2 lines
        //   StudentESA *soa;  // Ptr to Orderable Array of Student ptrs
        //  Student **sa; // Ptr to actual array of Student ptrs

            // These might be useful;
        unsigned int top; // Index value 1 above highest OA element used
        unsigned int btm; // Index value 1 below lowest OA element used
  public:
        StudentDQI (unsigned int); // Create a DQ with this initial size
        StudentDQI (StudentDQI&); // Equate this to an existing DQ (can use private parts
        directly)
        ~StudentDQI(); // Destruct the DQI (free OA space)

       int getSize ();  // Gets # elements (Student *) in the DQ
      bool isEmpty(); // True if no elements held ("pop" will fail)

     int pushFront (Student &); // Prepend a new element to the front
    Student & popFront ();      // Remove the first element and return it
    Student& lookFront ();      // Return the first element but do not remove it

     int pushBack (Student &); // Append a new element to the back
    Student & popBack ();     // Remove the last element and return it
    Student& lookBack ();     // Return the last element but do not remove it
};
```

Homework 2 Assignment Logistics

There are 9 files involved in Homework 2.

4 you already have
3 that I will supply
2 that you must return.

Files you already have

1. Student.h: Defines the Student object (data and code) whose pointers are being collected.
2. StudentEsa.h: Defines the Orderable Array interface.

   // Not required if you use arrays
3. StudentEsa.cpp. Implementation of StudentESA.h. **// (**This was what you submitted in HM 1).

   // Not required if you use arrays
4. HM2Instructions.pdf: This file which describes the HM 2 assignment

Files I will supply

1. StudentDQI,h: Defines the Deque interface. <u>You may add additional private data members</u>
2. Hm2Test.txt - The data that will drive the testing. **Supplied**
3. CIS22C-HM2.cpp - The "mainline" which reads those test commands and invokes Deque calls.

Printouts you will return

1. **StudentDQI.cpp:** DQ Implementation of StudentDQI.h (optionally wrapping StudentESA)**.**
2. **Hm2Output.txt:** The collected output from a successful test (capture **cout**)

Assignment Logistics:

1. Copy the 4 files you have and the 3 newly supplied files into a directory / folder / Visual Studio project (or equivalent).

2. Add your code in StudentDQI.cpp to implement the "interface contract" defined in StudentDQI.h

3. Optionally add any private data to StudentDQI.h that your StudentDQI.ccp implementation requires

4. Build the program.

5. Run the CIS22C-HM2 program, specifying " Hm2Test.txt " as an argument

6. Record the output.

7. **Debug:** If errors, go back to step 2

8. Print out and return

Goal: We would like the DQ use the ESA in such a way as to avoid the massive data shifts required to support push front and pop front.

<Here is the suggested solution. If you think you have a better one, feel free to program it (explaining the details of your alternative in the comments)>.

This solution requires the following **use of the ESA located within the DQ class.**

<Note: If you implemented the DQ class on top of a standard array, what follows is only a guide rather than a set of detailed instructions.>

Implementation Details

You need to position the Deque in the middle of a larger ESA array

- The DQ constructor has an argument "N" specifying the max depth of the DQ
  - Allocate an ESA array of twice that size (2N)
  - Set "*below the bottom*" (B) and "*above the top*" (T) private int variables to midpoint of ESA array (N and N+1 respectively)
  - **Issue N consecutive "Append" commands to the ESA with a Null Student Ptr.** The DQ is now centered on index N. It can be pushed from both the top or bottom using the ESA append and insert commands, or popped from the top or bottom using the ESA remove command.

- From this point on:
  - Any necessary resizing of the array is done by the ESA
  - If "top" and "bottom" ever differ by 1, the DQ is empty, and "**pops**" fail
  - A DQ **push front** and **pop front** move the B index
  - A DQ **push back** and **pop back** move the T index

<There is (as there usually is) one degenerate case where performance gets ugly. That is where there have been far more **push fronts** issued to the DQ than there have been **pop fronts**. In this case, the "bottom" pointer has reached ESA index 0, and a **push front** request arrives. We can't issue a request to ESA "store" at index -1, so all the elements have to be moved back one. Not good.>

This homework does NOT require you to deal with this situation … the value of N was set by the creator of the DQ, and we will assume they know what they are doing.