CS 201 Data Structures Library Phase 1 Due 2/15

For phase 1 of the CS201 programming project, we will start with a circular dynamic array class and extend it to implement some of the algorithms discussed in class.

Your dynamic array class should be called CDA for circular dynamic array. The CDA class should manage the storage of an array that can grow and shrink. The class should be implemented using C++ templates. As items are added and removed from both the front and the end of the array, the items will always be referenced using indices 0…size-1. Your CDA class should include a flag indicating whether the array is in forward or reverse. The forward and reverse should not be confused with sorted order.

A = { 1 , 3, 4, 2} => Forward = { 1, 3, 4, 2 } => Reverse = { 2, 4, 3, 1 }

The **only include files** that should be necessary for the CDA class would be related to generating random values for Quickselect.

The public methods of your class should include the following (elmtype indicates the type from the template):

Function	Description	Runtime
CDA();	Default Constructor. The array should be of capacity 1, size 0 and ordered is false.	O(1)
CDA(int s);	For this constructor the array should be of **capacity** and **size s**.	O(1)
~CDA();	Destructor for the class.	O(1)
elmtype& operator[](int i);	Traditional [] operator. Should print a message if i is out of bounds (outside 0…size-1) and return a reference to value of type elmtype stored in the class for this purpose.	O(1)
void AddEnd(elmtype v);	increases the size of the array by 1 and stores v at the end of the array. Should double the capacity when the new element doesn't fit. The new element should be accessed by the user at array[size-1].	O(1) amortized
void AddFront(elmtype v);	increases the size of the array by 1 and stores v at the beginning of the array. Should double the capacity when the new element doesn't fit. The new element should be accessed by the user at array[0].	O(1) amortized
void DelEnd();	reduces the size of the array by 1 at the end. Should shrink the capacity when only 25% of the array is in use after the delete. The capacity should never go below 4.	O(1) amortized
void DelFront();	reduces the size of the array by 1 at the beginning of the array. Should shrink the capacity when only 25% of the array is in use after the delete. The capacity should never go below 4.	O(1) amortized
int Length();	returns the size of the array.	O(1)
int Capacity();	returns the capacity of the array.	O(1)
void Clear();	Frees any space currently used and starts over with an array of capacity 4 and size 0.	O(1)

void Reverse();	Change the logical direction of the array. This should be accomplished using a Boolean direction flag, simply copying the array in the reverse order will take O(n) time which will be too long.	O(1)
Elmtype Select(int k);	Array is not sorted. Perform a quickselect algorithm to return the kth smallest element. Quickselect should choose a random partition element.	O(size) expected
void Sort();	Sorts the values in the array using an O(n lg n) sorting algorithm of your choice.	O(size lg size) worst case
Int Search(elmtype e)	Array is not sorted, perform a linear search of the array looking for the item e. Returns the index of the item if found. Otherwise, return -1.	O(size)
int BinSearch(elmtype e)	Array is sorted, perform a binary search of the array looking for the item e. Returns the index of the item if found. Otherwise, return a negative number that is the bitwise complement of the index of the next element that is larger than item e or, if there is no larger element, the bitwise complement of size.	O(lg size)

Your class should include proper memory management, including a destructor, a copy constructor, and a copy assignment operator.

For submission, all the class code should be in a file named CDA.cpp. Create a makefile for the project that compiles the file Phase1Main.cpp and creates an executable named **Phase1**. A sample makefile is available on Blackboard. **Only the compiler flag should be changed in the makefile**. Place both CDA.cpp and makefile into a zip file and upload the file to Blackboard.

- ☐ Create your CDA class
- ☐ Modify the makefile to work for your code (changing compiler flags is all that is necessary)
- ☐ Test your CDA class with the sample main provided on the cs-intro server
- ☐ Make sure your executable is named **Phase1**
- ☐ Develop additional test cases with different types, and larger arrays
- ☐ Create the zip file with **CDA.cpp and makefile**
- ☐ Upload your zip file to Blackboard

No late submissions will be accepted. There will be an opportunity to resubmit by 2/28 Resubmissions will have a 20 point penalty.