In this lab, you will develop a class that supports array rotation. Rotating an array is an operation where you shift all elements of the array some number of positions left or right, and elements that are shifted off of the left or right end of the array "wrap around" to the right or left end, respectively.

For example, if we rotate the array [1, 2, 3, 4, 5] to the right by 1, we get the array [5, 1, 2, 3, 4]. If we rotate it to the left by 2 (or to the right by 3), we get [3, 4, 5, 1, 2].

Your rotate function only needs to accept arguments in the range -(size - 1) to (size - 1), with negative values indicating a left rotation and positive values a right rotation, though it is reasonably easy to support larger values using modular arithmetic. You may choose to implement the left and right rotations using two different cases. Also, I would recommend allocating a temporary array to perform the rotation; just make sure to free memory appropriately afterwards. You may wish to use the memcpy function (in <cstring>, see documentation) to copy array elements from one place to another, but for loops will work, as well.

Your class should also support adding an arbitrary number of elements to the array via push\_back, similar to a vector. To accomplish this, your class should have data members that store the size (number of elements added) and capacity (allocated size) of the array. When pushing back another element, if size == capacity, you should allocate an array twice as large as the previous array (and update the capacity) and copy the elements in the previous array to the new one, before freeing the old array and updating the pointer data member to point to the new array. At this point, you can add the new element as normal.

Your class must also support appropriate copy and move operations, as well as a destructor. A full list of required functions and their descriptions appears below:

* RotatableArray(): creates an empty array with capacity = 8
* RotatableArray(const RotatableArray&): copy constructor; makes a deep copy of its argument
* RotatableArray(RotatableArray&&): move constructor; takes the array from its argument
* RotatableArray& operator=(const RotatableArray&): copy assignment; makes a deep copy of its argument
* RotatableArray& operator=(RotatableArray&&): move assignment; takes the array from its argument
* ~RotatableArray(): destructor; deallocates the array
* void push\_back(int): adds the given integer to the back of the array, doubling the capacity if out of space
* int pop\_back(): removes an element from the array and returns it. If array is empty, no change and returns 0.
* int& operator[](int): returns a reference to the given element of the array (no bounds checking required)
* int operator[](int) const: similar to previous, as an accessor
* void rotate(int): rotates the array; positive arguments rotate to the right, while negative arguments rotate to the left. You may assume the argument is in the range -(size - 1) to size - 1.
* ostream& operator<<(ostream&, const RotatableArray&) **\*\*Not a member function\*\***: prints the elements of the array, separated by spaces. You may print an extra space at the end for convenience

You should submit a single source (.cpp) file containing your solution to this assignment. Your code must be able to compile using g++. Submissions that do not compile when using g++ will receive a substantial penalty.