CS 260
Programming Lab 1

For this lab you will be creating a class that implements an enhanced array class. This should be a review of arrays, classes, and exceptions.

This lab is worth a total of 100 points, 10 points are the self-evaluation, 60 points for the basic lab as specified below, 20 points for the advanced version of the lab, and 10 points for a solution to the thinking problem.

Base Lab Specification

Create a class called ArrayInt. This class will need to work with the provided driver, so class names and methods must match this specification. Your implementation should use simple arrays, not lists or vectors. (Python programmers use either the array module or pseudo arrays as described in Moodle).

Your class will need to have the following methods. Note that the driver is set up to test the methods incrementally in the following order.

- Constructor (Python init) – create an array of ***n*** integers where ***n*** is the parameter passed in. If no value is passed in, create an array of 10 items (default constructor). If ***n*** is less than 1, create an array of size 10. ***Do not initialize the elements of the array.*** (Python requires initializing the array, use zero).
- int getSize() – return the current size of the array
- void resize(int size) – resize the array to the new **size**, copying values as needed. If the new size is less than or equal to the current size, do nothing. You are not to use built in resize methods in C# or Python.
- void append(int value) – add ***value*** at the next available location in the array. If the next location is greater than or equal to the current size of the array, create a new array of size*2 and copy all values to it. The first value should be added at index 0 and so forth. This will require keeping track of the next available location.
- int getLast() – return the value at the previously available location. If the array is empty, throw an ***out_of_range*** exception (Python use ***IndexError***).
- void deleteLast() – similar to getLast, but deletes the last item and does not return anything. Note that deleting an item means making it not available. This does not require setting it to a default value or doing a C++ delete. If the array is empty, do nothing.
- string listElements() – return a string containing a comma-separated list of the elements in the array from location zero through the largest location containing valid data. If the array is empty, return the string "Empty Array".
- void insertAt(int index, int value) – insert ***value*** at ***index*** location, shifting all values at that location or higher up one. If this shift would cause elements to be stored past the end of the array, create a new array of size * 2 and copy all values to it. If index is less than 0, greater than the largest location containing valid data, or greater than the array size, throw an ***out_of_range*** (***IndexError***) exception.

- int removeAt(int index) – save the value at index location, shifting all values at that location or higher down one position. Return the saved value. If index is less than 0, greater than the largest location containing valid data, or greater than the array size, throw an ***out_of_range***(***IndexError***) exception
- bool find(int value) – return true if value is in the array, false otherwise.
- bool findRemove(int value) – if value is in the array, remove it while maintaining the same array order. Return true if value is in the array, false otherwise.
- int findLargest() – return the largest element in the array. If the array is empty, throw an ***out_of_range*** exception (***IndexError***).
- void removeLargest() – remove the largest element in the array, while maintaining the same array order. If the array is empty, throw an ***out_of_range*** exception (***IndexError***)

Advanced Lab Specification

Complete all requirements of the basic lab. Add the following methods. Note that the previous methods are maintaining a compact or dense array where all indexes less than the next location to store a value contain valid data. These two methods allow storing data randomly in the array and might cause gaps in the valid data.

- int getAt(int index) – return the value at **index** location. If **index** is less than 0 or greater than the array size, throw an ***out_of_range*** exception (***IndexError***). This may return a value from a location that was not previously set.
- void setAt(int index, int value) – store **value** at **index** location. If **index** is less than 0 or greater than the array **size**, throw an ***out_of_range*** exception (***IndexError***). Make sure that any following appends will not overwrite this value. This might change a value that was previously set or it might store the value at a location at an index higher than any previous append or insert has used.

Thinking problem

Here is a set of numbers. Using your base array methods, list the numbers from largest to smallest.

Testing and requirements

Use the supplied driver in Moodle to test your class and show that it works properly. (Note that there are drivers supplied for C++ and Python. If you are using another language, please provide an equivalent console driver.) The drivers are set up to test your methods in an incremental manner, it is suggested that you get one set of methods working before starting on the next set.

Note that you should not initialize your array to some value such as zero and when you remove a value you need to shift the other values in the array down, not try to store some special value there.

You should use the resize method whenever you have to increase the size of the array rather than having duplicate code. Likewise, you should look at the functions that are required and see where you can use an existing function rather than writing new code.

Note that this project does not make sense in that you would normally not have a single data structure that uses both setAt and append— typically you use one or the other method to add new data. The requirements are this way so that you have to think about the implications of using conflicting access methods at the same time.