BRONX COMMUNITY COLLEGE
of the City University of New York
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

CSI 33 Section E01 **Project 4**
Fall 2017 **November 6, 2017**
Due: Monday, November 13, 2017

Programming Project Number 4: Enhancement of the List Class

This is Programming Exercise 1 on page 400 of the text. The code for the `List` class provided in chapter 10 is the starting place. (For this project, a `List` is a container of integer values only. And for some reason the exercise says to include methods that are already in the code—you don't have to redo these!) The methods to be added will give the `List` class some more of the behavior of the `list` class in Python. These methods will be `extend`, `index`, `insert`, `pop`, and `remove`. Already added for your use is a friend function `operator<<(ostream &o, List &l)` which will send a display form of the list to an `ostream` such as `cout` and return a reference to the same ostream object that it takes as its first formal parameter. This will give the same appearance to a `List` as a Python `list`: brackets containing the item values, separated by commas, for example, `[1, 2, 3, 4]`. Also added is a friend function, `operator>>(istream &o, List &l)`, to perform input of a `List` object in the usual format (brackets around comma-separated integers), to use when testing the `expand` and `operator+=` methods.

Method details

To remind you what these functions do in Python:
The method `extend(const List &l)` will add the entire contents of a second `List` to a `List` object. It is actually already written, with one line which calls `operator+=`, which is the one you must actually write.

The method `index(int value)` will return the position (index) of the first occurrence of `value` in the `List` object. A precondition is that the value must be somewhere in the `List`.

The method `insert(size_t i, int value)` will insert `value` at position `i`. If `i` is greater than the size of the `List` then `value` will be appended to the end of the `List`.

The method `pop(size_t i)` will remove the item at position `i`. A precondition is that `i` is not greater than the size of the `List`. The default value for `i` will be one less than the size of the `List`, that is, the index of the last value, which will then be removed and returned.

The method `remove(int value)` will remove the first occurrence of `value` in the `List`. A precondition is that the value must be somewhere in the `List`.

Some Hints

Many methods will require for loops ranging through the index values of the `_data` array. If you are looking for a certain value, and find it, use `break;` to exit the loop, as you would in Python.

The test program is written to handle exceptions which are `thrown`, the same way exceptions are `raised` in python. An example of how to do this has been left in the code—so use exception handling to guarantee appropriate preconditions.

If `List` objects must be expanded, follow the book's advice and double the current capacity by copying the entire array into one that is twice as big. In other words, use the `resize()` method, which is already there, with a large enough parameter value. Remember that, even with the copying, `append` is still, on average, $\Theta(1)$—the time does not significantly depend on the size of the array.

The test program has already been wired to test the methods you will write, so you don't have to change `TestList.cpp` at all. Just add the code into the methods assigned—they currently do nothing.