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

CSI 33 Section E01 **Project 3**
Fall 2017 **October 2, 2017**
Due: Monday, October 16, 2017

Programming Project Number 3: Implementation of Stack and Queue Classes

Stack ADT

You are to define a `Stack` class which implements the Stack ADT, that is, it will have the methods `push`, `pop`, `top`, `size`, and `__init__`. You should also implement a unit-testing file for this class.

Queue ADT

You are to define a `Queue` class which implements the Queue ADT, that is, it will have the methods `enqueue`, `dequeue`, `front`, `size`, and `__init__`. You should also implement a unit-testing file for this class. The two classes can be tested together by importing them into the palindrome.py file provided with Chapter 5.

Implementation Using Linked Structures

Both classes, `Stack` and `Queue`, will be implemented using a linked data strategy. That is, you will use `ListNode` objects from the class of that name defined in Chapter 4. You should import the file `ListNode.py` into your project for that purpose. Warning: you will not receive credit if you define your `Stack` (or `Queue`) class using the `LList` class defined in the text, even though it uses `ListNodes`—it has the same interface as a Python `List`, so you would not be creating any new code. You must use the `ListNode` class directly in your implementations. The hints in the next section give you ways you can still use `ListNodes` using coding ideas from `LList` class methods.

Hints on Implementation Using Linked Structures

A `Stack` is accessible at one end only, its top. This corresponds exactly to the head attribute of a linked list. Pushing and popping is then just adding a node to or removing a node from the head of a linked list. You can use code from the `LList` class of Chapter 4, making any necessary modifications. For example, if the `LList` class has an attribute called `head`, then for the `Stack` class, change the code so that the attribute is called `top`.

A `Queue` must be accessed from either of two ends, its front or its back. Thre linked structure for a queue will therefore need two attributes, `_front` and `_back`. (Follow the philosophy of thinking all ADT attributes as private, using an underscore as the first character. In languages like C++, this will indicate privacy which can be enforced.) The dequeue method should be implemented exactly like the `pop` method of the `Stack` class, removing a node from the front of the `Queue`. The `enqueue` method will be more complicated, since it must add a new `ListNode` object to the back of the `Queue`.

One more warning about implementing the `Queue` class: there is already a `Queue` module in the Python library, so if you try to implement your `Queue` class in a file called `Queue.py` and then try to say `from Queue import Queue` it will not use your file. You must call your file `MyQueue.py` and you must say `from MyQueue import Queue`. Chapter 5 shows an example of using the `Queue` class which does exactly this, in `palindrome.py`.

Finally, in exploring programming techniques using linked structures, always draw box-and-arrow diagrams to illustrate the Python statements of the methods you write, as we have done in class and in the text.