Assignment 2

(Due Friday of week 5 - Apr 26)

1. Write separate programs for the following exercises in Java, Python, or C#. **Each file should have your name at the top in comment, with short description of what is implemented in that file.** Make sure your files have appropriate names as indicated in each exercise. Programs should write output to the Console and have hard coded input in main.

   Note 1: If a program is not in approved programming language (or in different language than previous assignment) or has any syntax error, no points will be awarded for that exercise

   Note 2: Submitting wrong files or in the wrong format or corrupted files will not be accepted nor will any re-submission be allowed for any such mistake. It is your responsibility to submit the correct files.

   a) **DLinkedListADT:** Implement doubly linked list ADT using the **pseudo code** covered in class. You must have all the operations in the pseudo code with the SAME naming convention and statements. Any operation that is implemented differently from the one given, will not earn credit. Write test program with main method creating an instance of the DLinkedListADT and calling all the operations to demonstrate they work correctly. You must implement a class called DLinkedListADT for the ADT implementation in one file and a **separat**e TestDLinkedList class for the test in a different file.

   b) **ConvertToArray:** Implement a method that takes DLinkedListADT instance of size n and two integers x and y as parameters. The method then allocates an array **numbers[x][y]** and copies all elements over from the list to the array and returns the array to caller. The method must check if there are too many/too little elements and that x and y values are greater than 0 and throw an exception if not. Analyze time/space complexity of the method. Write a main method to test the code.

   For example:
   1. Given linked list with elements {1.0, 2.0, 3.0, 4.0, 5.0, 6.0 and x=2, y=3 we will have array numbers[2][3] and the contents:

	0	**1**	**2**
0	1.0	2.0	3.0
1	4.0	5.0	6.0

   2. Given linked list with elements {1.0, 2.0, 3.0, 4.0, 5.0, 6.0} and x=1, y=3 an exception will be thrown.

   3. Given linked list with elements {1.0, 2.0, 3.0, 4.0, 5.0, 6.0} and x=1, y=6 we will have array numbers[1][6] and the contents:

	0	**1**	**2**	**3**	**4**	**5**
0	1.0	2.0	3.0	4.0	5.0	6.0

c) **ListInsert:** Add new linked list operations to DLinkedListADT. The operation is called insertAfter and takes two keys as parameter: **insertAfter(double oldKey, double newKey)**. Then it finds the first node in the list that matches the oldKey and insert a new node with newKey value as the next node in the list. If list is null it just adds the new Node and if the oldKey is not found it adds to the end of the list. Add test cases to TestDLinkedList class from exercise (a) to demonstrate the code works. Analyze time/space complexity of the new operation.

   For example:
   Given linked list with elements {1.0, 2.0, 3.0, 4.0, 5.0, 6.0} when we call insertAfter(2.0, 8.0) the updated list will have nodes: {1.0, 2.0, 8.0, 3.0, 4.0, 5.0, 6.0}

d) **QueueADT**: Implement **Queue ADT** using the DLinkedListADT from previous exercise to include operations:
   - QueueADT(int max)
   - boolean isEmpty()
   - boolean isFull()
   - void enQueue(double item)
   - double deQueue()
   - int size()

   You must use the QueueADT pseudo code covered in class and use the given operation names and implementation. You cannot not have any other operations. Write test program with main method to demonstrate the correct working of **each** operation (use double values for your test data). You must implement a class called QueueADT and separate TestQueue class each in a separate file.

e) **StackADT**: Implement **StackADT** using the DLinkedListADT from previous exercise to include operations:
   - StackADT(int max)
   - boolean isEmpty()
   - boolean isFull()
   - void push(double item)
   - double pop()
   - int size()

   You must use the **pseudo code** covered in class and use the given operation names and implementation. You should not have any other operations. Write main method to demonstrate the correct working of **each** operation in a separate test program (use double values for your test data). You must implement a class called StackADT and TestStack class in a separate file.

f) **Stack1:** A letter means doing a push operation and an asterisk means doing a pop operation n the below sequence. Give the sequence of letters which are returned by the pop operations when this sequence of operations is performed on an initially empty stack.
   ABCDEF*GH***I*****

g) **Stack2:** Given an empty stack in which the values A, B, C, D, E are pushed on the stack in that order but can be popped at any time, give a sequence of push and pop operations which results in pop()ed order of **ADCEB**

2. Record a video(s) 10-20min long explaining the implementation and solution of each of the above programs to include showing the running program and output. You may have two separate videos if you need more time to explain all exercises. **DO NOT create separate videos for each exercise!!!**

3. **Submission instructions:** Submit one zip file with all the programs, separate pdf or word file with stack solutions problems f and g, and a separate zip/rar file(s) with the video(s).

Grading Rubric

NOTE: You will not earn any points for the implementation unless you submit the video which explains them.

Points	Criteria
10	Programs use object oriented program approach (e.g. class for Queue, Stack, etc.). Programs have the appropriate naming convention, author's name, and brief description of the implementation in **each file**
20	**DLinkedListADT:** Correctly implements DLinkedListADT class and its operations using pseudo code covered in class to include naming convention and the statements Correctly implements the test program TestDLinkedList which calls all operations in separate file to demonstrate correctness of the code Video correctly explains the ADT and test implementation, and shows and explains running program and output
20	**ConvertToArray:** Correctly implements a method that takes DLinkedListADT instance of size n and two integers x and y as parameters. There is correct analysis of time/space complexity of the method that shows all calculations There is a correct main method to test the code (both valid and some invalid data). Video explains the implementation of program and test and analysis and shows the running test program running
20	**ListInsert:** Correctly implemented and added new operations to DLinkedListADT - **insertAfter(double oldKey, double newKey)**. Correctly added test cases to TestDLinkedList class from exercise (a) to demonstrate the code works. There is correct analysis of time/space complexity for the new operation. Video explains the implementation, analysis, and shows program running

20	Correctly implements **QueueADT (using DLinkedListADT)** in single file using pseudo code covered in class. Names and implementation matches the pseudo code. Correctly implements **StackADT (using DLinkedListADT)** in single file using **DLinkedListADT** and pseudo code covered in class. Correctly implements test programs in separates files with given naming convention which demonstrate each operation Video correctly explains implementation and operations of each ADT and shows and explains the test programs running
10	**Stack 1 and Stack 2:** Correct solutions are given and explained in the video