National University of Singapore
School of Computing
IT5003: Data Structure and Algorithm
Semester I, 2019/2020

Tutorial x Lab 3
List ADT & Linked List

General idea: Tutorial type questions will be listed first in this document. Some tutorial question will then be implemented in the lab portion (later part of this document). Lab exercise cases to be submitted on Coursemology will be marked [SUBMISSION] at the front.

1. [Using List ADT] In scientific experiments, we frequently have to deal with data of the form (X, Y), i.e. a point on a 2-D plane. Let's designed a new **dataset** ADT to store and manipulate these data. Below are a few design decisions:
   - Each data point is represented by a **Python `tuple`** for simplicity, e.g. (1.23, -4.56).
   - The **`DataSet`** class has minimally the following two operations:

```
class DataSet():
    """
    DataSet ADT implementation using List ADT (any implementation)
    """
    def addPoint(self, newPoint ):
        """ Add a new point [newPoint] to the data set.
        [newPoint] should be a 2-tuple (x, y)
        """
        pass
    def findNearestPoint(self, refPoint ):
        """ Return the point nearest to [refPoint] in the data set.
        Return None if the data set is empty.
        """
        pass
```

   - In the **`DataSet`** class, we have decided to use **List ADT** as the internal data structure, i.e. we are using a simple ADT to build something more specialized. You can use either the **`ListArray`** or the **`ListLinkedList`** implementations.

   Questions to discuss:

   a. Give pseudo-code (or actual code) for the two operations and the constructor of the **`DataSet`** class.
   b. Is **`List`** ADT a good choice to implement the **`DataSet`** class in this case?
   c. Doe the specific version of **`List`** ADT implementation impacts the performance of **`DataSet`** class?

2. [Linked List Implementation of List ADT] Analyze the efficiency of linked list implementation of List ADT in term of time and space used.

3. [Variant of List ADT] Let's design and implement a **Sorted List ADT** using **Singly Linked List.** The final specification is very close to the normal List ADT (linked list implementation **ListLinkedList**), except for the **insert()** method:

```
class SortedList:
    """
    SortedList ADT implementation using linked list
    """
    def insert(self, newItem):
        """ Insert [newItem] in ascending order.
            Return True.
        """
    ... other methods not shown ...
```

You can see that the insertion index is not given for the **insert()** method. For sorted list, an item should be inserted to maintain ascending sorted order. E.g. given a list of {1, 4, 7, 8}, insertion of **3** will give {1, **3**, 4, 7, 8}. New item with the same value with existing item in the list will be inserted at the end. e.g. original list {1, 4, 4, 7, 8}, inserting another '4' will give {1, 4, 4, **4**, 7, 8}.

Write the insertion method. You can add other helper methods if needed.

4. [**Variations of Linked List**] Linked list, with its many variations, can be quite confusing. One way to understand the differences between the variations is to make a comparison table. Try to fill in the following table, the columns are:
   - **Maximum "hops"**: Which node in a **N-nodes** list is the hardest to reach? What is the number of "hops" required to reach this node?
   - **Memory space**: What is the memory requirement for N-nodes of list? You can assume that each item takes **X** amount of space and each pointer takes **Y** amount of space.

   You can assume all linked list variations has a **head pointer** pointing to the first node in the list **unless otherwise specified**.

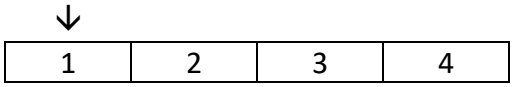
5. **[Doubly Linked List**) We have covered the general insertion and removal in doubly linked list in lecture. Let us now use it as an alternative implementation of the List ADT. Define a new class **ListDLL** as the new implementation class for List ADT. You are free to try out the complete implementation (could be helpful for your lab submission questions ☺). In the tutorial, we will only cover the following in details:
    - The **ListDLL** class constructor
    - The **insert**() method
    - Is doubly linked list helpful in List ADT? Why?

~~~ Lab Questions ~~~

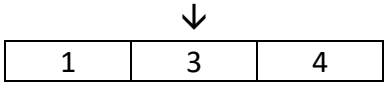
1. ***Josephine*** is a beautiful princess. She's so beautiful that there are ***N*** princes proposing to her (needless to say, ***N*** could be as large as 1,000!). After lots of consideration, Josephine and her parents realized that all these princes are perfect, so that it is impossible to choose one and reject the others with a good reason. They decided that Josephine would not reject the one chosen by Fate, while other princes could not ask for a fairer decision.

    The *N* princes, **numbered from 1 to *N***, are asked to stand in a circle. Our beautiful princess would choose a random positive number ***K*** and begin to find her fiancé with that number. Starting from the first person, she counted to ***K***, and removed that *K*-th person from the circle. Then, she **starts counting from the next person** and removes the *K*-th person. This process is repeated until all candidates are removed but ***The Chosen One***™.

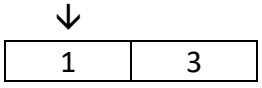
    For example, if N = 4 (four princes) and the random number K = 2:

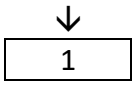
    The princes are represented by a number and the arrow refers to where we start the counting. The first prince to be removed is "2":

    Note that we start the next counting after the removed prince. The next prince removed is "4":

    Remember that they are standing in a circle. So, the counting will wrap around and start from the front. The next prince removed is "3":



Since there is only one prince left, prince "1" is the **chosen one**!

Everything has been prepared, except one minor problem. To be sure that the processes are random enough, Josephine chose a large number *K* (any positive number, can be much later than N itself!). As the result, counting and removing one princess after another would be a painful process. She leaves this work to you — the ***Royal Programmer***.

Read the **`Josephine.py`**, pay attention to the specification for each of the operation. **You have to use Circular Linked List as the internal structure of the Josephine class**. You can find additional test cases on Coursemology.

[Credit: ***Josephine*** is a very famous programming problem on circular linked list. The current write up was inherited from my predecessor in a SoC programming courses. I think the originator of the story has already been lost in history ☺.]

We will evaluate both correctness and programming style of your submitted code:

a. [Correctness 70%]: The code works according to the functional requirement, e.g. output value, output format, side effect, efficiency etc.

b. [Programming Style 30%]: Reuse own code whenever appropriate, modularity, good variable / method naming convention, comments (only if appropriate)