```
# CMPT 145 Course material
# Copyright (c) 2017-2021 Michael C Horsch
# All rights reserved.
#
# This document contains resources for homework assigned to students of
# CMPT 145 and shall not be distributed without permission.  Posting this
# file to a public or private website, or providing this file to a person
# not registered in CMPT 145, constitutes Academic Misconduct, according
# to the University of Saskatchewan Policy on Academic Misconduct.
#
# Synopsis:
#     Defines the List ADT
#


class node(object):
    """ A version of the node class with **public** attributes.
        This makes the use of node objects a bit more conveinient for
        implementing LList class.

        IMPORTANT: Since there are no setters and getters, we use the attributes
directly.

        This is safe because the node class is defined in this module.
        No one else will use this version of the class.
    """

    def __init__(self, data, next=None):
        """
        Create a new node for the given data.
        Pre-conditions:
            data:  Any data value to be stored in the node
            next:  Another node (or None, by default)
        """
        self.data = data
        self.next = next

    # Note: use the attributes directly; no setters or getters!


class LList(object):
    def __init__(self):
        """
        Purpose
            creates an empty LList object
        """
        self._size = 0     # how many elements in the stack
        self._head = None  # the node chain starts here; initially empty
        self._tail = None


    def is_empty(self):
        """
        Purpose
            Checks if the given LList object has no data in it
        Return:
            :return True if the LList object has no data, or False otherwise
        """
        return self.size() == 0
```

```python
    def size(self):
        """
        Purpose
            Returns the number of data values in the given LList object
        Return:
            :return The number of data values in the LList object
        """
        return self._size


    def add_to_front(self, value):
        """
        Purpose
            Insert value at the front of the LList object
        Preconditions:
            :param value:   a value of any kind
        Post-conditions:
            The LList object increases in size.
            The new value is at index 0.
            The values previously in the LList object appear after the new value.
        Return:
            :return None
        """
        anode = node(value, self._head)
        if self.is_empty():
            self._tail = anode
        self._head = anode
        self._size += 1


    def add_to_back(self, value):
        """
        Purpose
            Insert value at the end of the LList object
        Preconditions:
            :param value:   a value of any kind
        Post-conditions:
            The LList object increases in size.
            The new value is last in the LList object.
        Return:
            :return None
        """
        anode = node(value)
        if self.is_empty():
            self._head = anode
        else:
            self._tail.next = anode
        self._tail = anode
        self._size += 1


    def remove_from_front(self):
        """
        Purpose
            Removes and returns the first value
        Post-conditions:
            The LList object decreases in size.
            The returned value is no longer in in the LList object.
        Return:
            :return The pair (True, value) if self is not empty
            :return The pair (False, None) if self is empty
        """
        if self.is_empty():
            return False, None
        if self.size() == 1:
            self._tail = None
        anode = self._head
        self._head = anode.next
        self._size -= 1
        return True, anode.data


    def get_index_of_value(self, value):
        """
        Purpose
            Return the smallest index of the given value.
        Preconditions:
            :param value:   a value of any kind
        Post-conditions:
            none
        Return:
            :return True, index if the value appears in self
            :return False, None if the value does not appear in self
        """
        anode = self._head
        i = 0
        while anode is not None:
            if anode.data == value:
                return True, i
            anode = anode.next
            i += 1
        return False, None


    def value_is_in(self, value):
        """
        Purpose
            Check if the given value is in the LList object
        Preconditions:
            :param value:   a value of any kind
        Post-conditions:
            none
        Return:
            :return True if the value is in the LList object, False otherwise
            :return (False if the LList object is empty)
        """
        anode = self._head
        while anode is not None:
            if anode.data == value:
                return True
            anode = anode.next
        return False


    def retrieve_data_at_index(self, index):
        """
        Purpose
            Return the value stored at the index
        Preconditions:
            :param index:   a non-negative integer
        Post-conditions:
            none
        Return:
            :return (True, value) if value is stored at index and index is valid
            :return (False, None) if the index is not valid for the LList object
        """
        if index < 0 or index >= self.size():
            return False, None
        anode = self._head
        i = 0
        while anode is not None and i < index:
            anode = anode.next
            i += 1
        return True, anode.data


    def set_data_at_index(self, index, value):
        """
        Purpose
            Store value at the index
        Preconditions:
            :param value:   a value of any kind
            :param index:   a non-negative integer
        Post-conditions:
            The value stored at index changes to value
        Return:
            :return True if the index was valid, False otherwise
        """
        if index < 0 or index >= self.size():
            return False
        anode = self._head
        i = 0
        while anode is not None and i < index:
            anode = anode.next
            i += 1
        anode.data = value
        return True


    def remove_from_back(self):
        """
        Purpose
            Removes and returns the last value
        Post-conditions:
            The LList object decreases in size.
            The returned value is no longer in in the LList object.
        Return:
            :return The pair True, value if self is not empty
            :return The pair False, None if self is empty
        """
        if self.is_empty():
            return False, None

        if self.size() == 1:
            anode = self._head
            self._head = None
            self._tail = None
            self._size = 0
            return True, anode.data

        # general case
        aprev = self._head
        anode = aprev.next
        while anode.next is not None:
            aprev = anode
            anode = anode.next

        self._tail = aprev
        aprev.next = None
        self._size -= 1
        return True, anode.data


    def insert_value_at_index(self, value, index):
        """
        Purpose
            Insert value at index
        Preconditions:
            :param value:   a value of any kind
            :param index:   a valid index for the LList object
        Post-conditions:
            The LList object increases in size.
            The new value is at index.
            The values previously in the LList object at index or later appear
after the new value.
        Return:
            :return If the index is valid, insert_value_at_index returns True.
            :return If the index is not valid, insert_value_at_index returns False.
        """
        if index < 0 or index > self.size():
            return False

        if index == 0:
            self.add_to_front(value)
            return True

        if index == self.size():
            self.add_to_back(value)
            return True

        # general case
        anode = self._head
        i = 1
        while anode is not None and i < index:
            anode = anode.next
            i += 1
        new_node = node(value, anode.next)
        anode.next = new_node
        self._size += 1
        return True


    def delete_item_at_index(self, index):
        """
        Purpose
            Delete the value at index.
        Preconditions:
            :param index:   a non-negative integer
        Post-conditions:
            The LList object decreases in size if the index is valid
            The value at index is no longer in the LList object.
        Return:
            :return True if index was valid, False otherwise
        """
        if index < 0 or index >= self.size():
            return False

        if index == 0:
            self.remove_from_front()
            return True

        if index == self.size() - 1:
            self.remove_from_back()
            return True

        # general case
        anode = self._head
        aprev = None
        i = 0
        while anode is not None and i < index:
            aprev = anode
            anode = anode.next
            i += 1
        aprev.next = anode.next
        self._size -= 1
        return True
```