BRAC University

Department of Computer Science and Engineering

CSE 220: Data Structures

Assignment 02

Task 1

Implement a **MyOrderList** ADT that is a circular singly-linked-list-based ordered (sorted) list ADT.

Elements

The elements in an order list are Nodes consists of an integer type key (all keys are unique) and a reference to the next node.

Structure

The order list elements are stored in ascending order based on their keys. For each list element E, the element that precedes E has a key that is less than E's key and the element that follows E has a key that is greater than E's key. The next field of the Node containing the maximum key refers to the Node containing the smallest key.

At any point in time, one element in any nonempty list is marked using the list's **cursor**. You travel through the list using operations that change the position of the cursor.

[You are not allowed to use any class variable other than cursor.]

Constructors and their Helper Method

MyOrderList ( )

Pre-condition:

None.

Post-condition:

This is the default constructor of MyOrderList class. This constructor creates an empty ordered list.

Methods in the Interface

void insert (Node newElement)

Pre-condition:

None

Post-condition:

This method inserts newElement in its appropriate position within a list. If an element with the same key as newElement already exists in the list, then it concludes the key already exists and do not insert the key.

Node retrieve (int searchKey)

Pre-condition:

The list is not empty.

Post-condition:

It searches for the ordered list for the element with key searchKey. If the element is found in the list, then moves the cursor to the element and returns the node containing searchKey. Otherwise, does not move the cursor and returns null to indicate that searchKey is not found.

Node remove ( )

Pre-condition:

List is not empty.

Post-condition:

This method removes the element marked by the cursor from a list. If the resulting list is not empty, then moves the cursor to the element that followed the deleted element. If the deleted element was at the end of the list, then moves the cursor to the beginning of the list.

Node remove (int deleteKey)

Pre-condition:

List is not empty.

Post-condition:

Removes the element from a list that contains the deleteKey. If the resulting list is not empty, then moves the cursor to the element that followed the deleted element. If the deleted element was at the end of the list, then moves the cursor to the beginning of the list.

void clear ( )

Pre-condition:

The list in not empty.

Post-condition:

Removes all the elements from a list.

boolean isEmpty ( )

Pre-condition:

None.

Post-condition:

Returns true if a list is empty. Otherwise, returns false.

Node gotoBeginning ( )

Pre-condition:

None.

Post-condition:

If a list is not empty, then moves the cursor to the element at the beginning of the list (i.e., the node that contains the smallest key in the list) and returns cursor. An empty list returns null.

Node gotoEnd ( )

Pre-condition:

None.

Post-condition:

If a list is not empty, then moves the cursor to the element at the end of the list (i.e., the node that contains the greatest key in the list) and returns cursor. An empty list returns null.

Node gotoNext ( )

Pre-condition:

List is not empty.

Post-condition:

It moves the cursor to the next element of the cursor in the list and returns that node. An empty list returns null.

Node gotoPrior ( )

Pre-condition:

List is not empty.

Post-condition:

It moves the cursor to the previous element or the cursor in the list and returns that node. An empty list returns null.

Node getCursor ( )

Pre-condition:

List is not empty.

Post-condition:

Returns the node marked by the cursor. An empty list returns null.

void showList ( )

Pre-condition:

None.

Post-condition:

Outputs the keys of the elements of the order list. If the list is empty, outputs “Empty list”.