UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS212: Practical 1

Release: Monday 13 February 2017
Deadline: Friday 17 February 2017, 18:00

Instructions

Complete the tasks below. Certain classes have been provided for you in the *files* subfolder of the practical download. You have been given a main file which will test some code functionality, but it is by no means intended to provide extensive test coverage. You are encouraged to edit this file and test your code more thoroughly. Remember to test "corner" cases. Upload **only** the given source files with your changes in a zip archive before the deadline. Please comment your name **and** student number in at the top of each file.

Task 1: Doubly-Linked Lists [36]

In previous courses, you have encountered singly linked lists, where every node has a reference to the node that comes after it. Doubly linked lists are similar, but every node has a reference to the node that comes before it as well as the node that comes after it in the list. You are required to implement a **doubly linked** list that supports searching capabilities. We have provided the structure of the `DLList` class (where the "DL" stands for "Doubly Linked"). The class has two protected variables, `head` and `tail` which should always point to the head and tail of the list, respectively.

The `DLList` has two methods for printing the contents of the doubly linked list, namely `printList`. This method takes as parameter a single boolean *verbose*. If *verbose* is true, the method more detailed output than otherwise. You are encouraged to use both of these during testing, but do not modify them in any way.

The `DLList` class also contains the implementation of `DLNode`, a nested helper class which represents the nodes in the linked list. Additionally, you are provided with an `EmptyListException` class that should be thrown when certain operations are attempted on an empty list.

Your task is to implement the following methods:

`void addToHead(T value)` **(7)**
: Inserts an element at the head of the list.

`void addToTail(T value)` **(3)**
: Inserts an element to the tail of the list.

`void removeHead()` **(6)**
: Removes the element at the head of the list. If the list is empty, an `EmptyListException` exception must be thrown.

`void removeTail()` **(6)**
: Removes the element at the tail of the list. If the list is empty, an `EmptyListException` exception must be thrown.

`void remove(T value)` **(6)**
: Removes the node that stores an element equal to *value*. If the list is empty, an `EmptyListException` exception must be thrown. If the element could not be found, the function should simply return.

`boolean isEmpty()` **(1)**
Returns true if the list contains zero elements, and false otherwise.

`int getCount()` **(1)**
Returns the amount of elements in the list. An empty list contains zero elements.

`int getPosition(T value)` **(6)**
Find the index of the first occurrence of *value* in the list. For example, if the head node contains *value*, the function will return 0. If the tail node is the first node to contain *value* and the list contains n elements in total, then the function will return $n - 1$. If *value* is not in the list, the function must return -1.

Only implement the functions listed above. Do not modify any other code that you were given.

Task 2: Self-Organising Lists [14]

Searching within a linked list has the drawback of requiring one to internally iterate through the list until the desired element is located. Self-organising lists can improve the performance of searches by re-organising elements as they are accessed. `SOList` (where "SO" stands for "Self-Organizing") inherits from `DLList`. Implement the new `access` function so that the list obeys the ***Move-To-Front*** self-organising strategy: each time an element is accessed using the `access` method, the element must be moved to the front of the list, unless it is already at the front of the list. If an element is not found, including if the list is empty, the element should be added to the tail of the list. You may add `private` helper functions to the class.

Submission

Submit your source files on the CS Website. Place all the files in a zip archive named as uXXXXXXXX.zip where XXXXXXXX is your student number. You have all week to finish this practical, regardless of what practical session you attend. Upload your archive to the *Prac1* slot on the CS website. Submit your work before the deadline. No late submissions will be accepted.