ALGORITHMS AND DATA STRUCTURES 2

TOPIC 7: LISTS
FORMATIVE EXERCISE

In this activity, you will work **individually** to implement a sorted single linked list.

PART 1: YOUR TASK

In this formative exercise, your task is to implement a sorted single linked list. The elements stored in the list are integers and they must be sorted from smallest to largest (the smallest element is in the head of the list).

This exercise will be automatically graded. Thus, you are provided with two files: LinkedList.cpp and LinkedList.hpp.

Your task is to complete the code that makes the methods in that skeleton code operative. **You must not change the prototypes of the methods**. You can add other methods and variables if you want.

PART 2: THE METHODS

Please, look at the content of the skeleton files you are provided with for this formative exercise (LinkedList.cpp and LinkedList.hpp). You can see there are six methods you must implement in the .cpp file:

- LinkedList: This constructor initialises the head of the list to NULL.

- ~LinkedList: This destructor deletes the list.

- insertSorted(ptr head, x): This method inserts the element x in its correct position in the list pointed by head. Given that the list must be sorted (from smallest to largest value) at all times, every time a new element is inserted into the list, it must be located in its correct position so the list remains sorted.

- length(head): This method returns the number of elements of the list pointed by *head*.

- search(head,x): This method returns the position where element *x* is in the list pointed by *head*. If there is more than one element equal to *x*, it returns the position of the first element. If the element is not in the list, it returns -1.

- remove(head,x): This method removes element *x* from the list pointed by *head*. If there is more than one element equal to *x*, only the first element is removed.

Two methods in the .cpp are already implemented to facilitate the debugging of your code:

- display(head): This method prints on screen the content of the list pointed by *head*.
- getHead(): This method returns the pointer to the list. This is necessary because, unlike the Hash Table submission (where *buckets* was public), *head* has been declared private in this implementation.

PART 3: SUBMISSION

When you want to submit your work, please compress both files (.cpp and .hpp) into a zip file. The compressed file can have any name.

In the My submission tab, press the blue 'Create Submission' button. Below the text 'Upload Files and Submit', press the blue 'Linked Lists' button, select the file and click 'Submit'.

Upon submitting, you will see the date of the submission in grey. It might take a while for the system to grade your code. You can wait for the grade or log out and come back later.

Once the grade is ready, you can click on the submission tab and then on 'Show grader output' to check on eventual errors.

You can upload your submission as many times as you want. Your highest score will be considered.

You can get familiar with the system by simply uploading the skeleton code provided for you. Naturally, all tests will fail in this case.

Next, attempt to implement the methods, one by one. After each step, run the tests once more, and check that the number of failing tests has decreased.