ALGORITHMS & DATA STRUCTURES
L.Ouarbnya
TOPIC 6: TREES
LAB WORK

In this activity, you will work **individually** implementing different functions for a binary search tree. Remember that you can discuss ideas and questions with your classmates, but you cannot see other's work neither others can see your individual work.

After finishing the implementation of your functions, you must answer a Lab Work Quiz that will test your knowledge on the understanding of the code you worked on.

WEEK 1: Tree operations, Part 1

Download the skeleton code[1] from learn.gold and do the activities described below.

Notice that these activities consider that you work 10 hours a week for this module: 2 hours for attending the weekly lecture, 2-3 hours of personal study (watching videos and understanding the concepts presented in them) and 5-6 hours of practical work (2 of them, in the lab). Hence, they are not meant to be totally finished during the lab session.

Activity 1: Have a good read at the code and identify the main blocks:
- how a tree node is defined
- how the root of the tree is defined
- how the insertion of a new node is carried out in a BST
- how pre-order traversal is used to print the content of the BST. Notice that there are some non-essential helper functions for making the printed nodes a bit more readable. You don´t need to use them if you don´t want to.
- how the main function allows you to interact with the user

Activity 2: Execute the code and make sure you know how to use it (at user level). The code starts with a BST already constructed. You can, of course, change this to start with an empty tree and insert your own nodes.
Although the traversals In-Order and Post-Order and the function remove() are already included in the menu, they are not implemented yet (well, actually they were implemented; we just deleted them).

[1] You can select one of two options: having a single .cpp/.java file or having multiple files. The second option is most used in professional environments, as it helps organise the information in shorter files. At this stage, you can choose any of them.

Activity 3: Implement the function inOrder() that displays on screen the content of the BST following the In-Order traversal. You can use the helper functions if you liked the way they print the information on screen. Otherwise, you can simply print the nodes one after another. You will find the prototype of this function in the code.

Activity 4: Implement the function postOrder() that display on screen the content of the BST following the Post-Order traversal. You can use the helper functions if you liked the way they print the information on screen. Otherwise, you can simply print the nodes one after another. You will find the prototype of this function in the code.

Activity 5: Implement the function search(x) that returns TRUE if the number x is in the BST and FALSE otherwise. Modify the main menu to include this function in the options. You will not find the prototype of this function in the code. You must build it from scratch.

Activity 6: Implement the function sum() that returns the value of the sum of all nodes in the BST. Modify the main menu to include this function in the options. You will not find the prototype of this function in the code. You must build it from scratch.

Activity 7: Calculate the time complexity of every function implemented (Activity 3-7). Use Theta notation for this.

WEEK 2: Tree operations, Part 2

Implement the function remove(x) that deletes the first occurrence of number x in a BST. Remember that remove() must take care of three different situations. You must implement the helper function getRMin() that gets the leftmost child of the right node. You will find the prototype of these two functions in the code.