ENSC 251 Lab Assignment 3

Lab Assignment Overview:

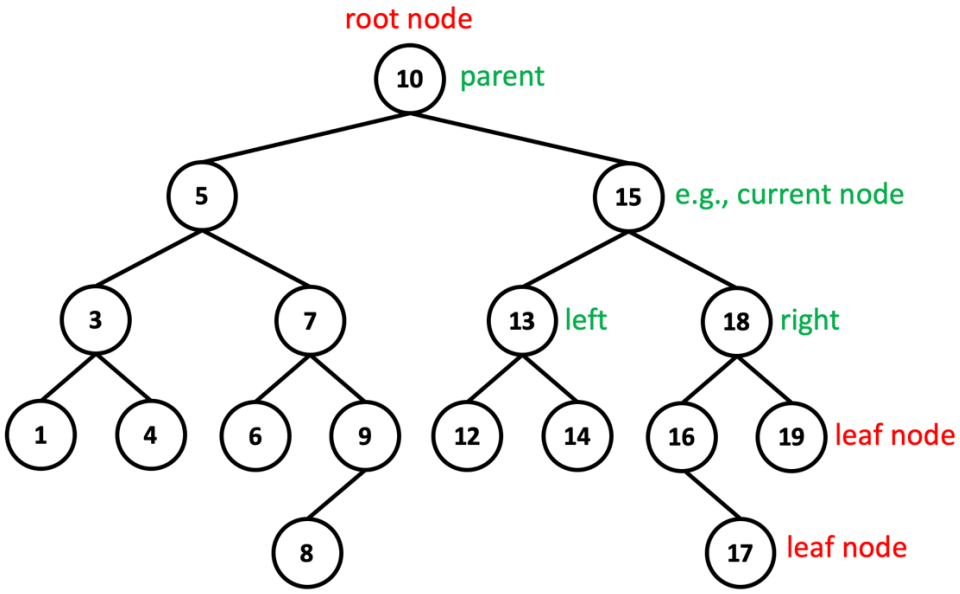
In the ENSC 251 course, you will work on four lab assignments using the skills learned throughout this course, i.e., Object-Oriented Programming (OOP) with C++. Each lab assignment weighs 10 marks. All lab assignments will be carried out and evaluated in pairs (i.e., two students per group) - I hope by now all of you have already found a partner. Some general grading logistics have been posted on our course website: https://canvas.sfu.ca/courses/70216/pages/lab-logistics.

Please make sure your code can compile and run correctly on the lab computer. If your code cannot compile, then your group can get at most 5 marks. If your code can compile, but cannot run, then your group can get at most 6 marks.

You can divide the work roughly based on the listed tasks; the overall marks for major tasks are also listed for your guidance, but not the detailed marks for every single item. Please include a simple document in each lab (include a txt or pdf file in the .zip file for submission) indicating which parts each member finished. There is some negative marking if you don't complete the listed tasks; details are not listed. Basically, the detailed grading scheme (except the overall marks for major tasks) for each lab will NOT be released before your lab grading is done. Think about you are in a real interview, nobody will tell you what the detailed grading schemes are.

Lab Assignment 3:

In this lab assignment, you will implement a Binary Search Tree (BST). A BST is a sorted binary tree, where one can efficiently search for any key using the binary search algorithm. An example BST is shown in the figure below.



The BST can be accessed from the *root node*, i.e., node 10 (the node with key=10) in this example. For each node, it has the following members: 1) key, which denotes the key value (*int* type) that the node stores; 2) a pointer to its left subtree; 3) a pointer to its right subtree; 3) a pointer to its

parent node. For example, for the node 15, its key value is 15, its left pointer points to node 13, its right pointer points to node 18, and its parent pointer points to node 10.

The *root node* is the entry node to the BST and its parent=NULL. When both the left and right pointers of a node point to NULL, that node is called a *leaf node*. For example, nodes 4, 6, 17, and 19 are leaf nodes. For some nodes like node 16, it may have the left pointer points to NULL and the right pointer points to another node (node 17 in this case); similarly, for some nodes like node 9, it may have the right pointer points to NULL and the left pointer points to another node (node 8 in this case). These nodes (i.e., nodes 16 and 9) are called *single child node*.

As shown in the figure, each node in the BST has the following properties:

- All nodes in its left subtree (if exist) have a key that is smaller than this node's key
- All nodes in its right subtree (if exist) have a key that is greater than this node's key
- There are no duplicate keys in the BST

According to the previous description, the type for each node can be defined as follows.

```
struct Node {
        int key; // key value of the node
        struct Node *parent; // pointer to parent node
        struct Node *left; // pointer to left subtree
        struct Node *right; // pointer to right subtree
};
typedef struct Node* NodePtr;
```

Specifically, in lab assignment 3, you will implement and test the BinarySearchTree class by building the following items. Assume all key values are positive integers in this assignment.

1) **[8.5 marks] Implement the *BinarySearchTree* class.** The BinarySearchTree class should have a member variable ***NodePtr root***, which points to the root of the binary search tree (BST). Implement the following **member functions**:
    a. [1 mark] Constructor functions and get function. In this case, no set function is needed to set the root member variable.
    b. [1.5 marks] The dynamic big three for this class since it has a pointer member variable, i.e., destructor, copy constructor, and overloaded assignment operator.
    c. [1 mark] **NodePtr searchNode(int key);** If the specified ***key*** is in the current BST, return that node; otherwise, return NULL.
    d. [1 mark] **NodePtr searchSuccessor(int key);** If the specified ***key*** is not in the current BST or is the largest in the current BST, print out a warning message and return NULL. Otherwise if the specified ***key*** is in the current BST, return the immediate successor node, whose key value is the smallest number that is bigger than the specified ***key***. In our example BST, searchSuccessor(2) returns NULL, searchSuccessor(19) returns NULL, searchSuccessor(7) returns node 8, searchSuccessor(15) returns node 16, searchSuccessor(10) returns node 12, searchSuccessor(8) returns node 9, searchSuccessor(4) returns node 5, etc.

e. [1 mark] **bool insertNode(int key);** If the specified ***key*** is in the current BST, e.g., when a user wants to insert key=15 for the example BST, print out a warning message and return false. Otherwise if the specified ***key*** is not in the current BST, you create a new node with the specified ***key***, and insert the node into the current BST, and return true. When inserting the new node, you should maintain the sorted property of the BST. For example, when a user wants to insert key=2 in the example BST, it should be inserted as the right child of node 1; when a user wants to insert key=11 in the example BST, it should be inserted as the left child of node 12.

f. [2 marks] **bool deleteNode(int key);** If the specified ***key*** is not in the current BST, print out a warning message and return false. Otherwise, if the specified ***key*** is in the current BST, delete this node (including the operation of free its memory), and return true. Moreover, you may have to adjust the BST to maintain the sorted property of the BST. For example, when a user wants to delete key=17 (leaf node) in the example BST, you simply delete node 17. When a user wants to delete key=16 (single child node) in the example BST, you have to copy its child node (node 17, including all its key and pointers) to this node (node 16), and delete the child node (node 17). When a user wants to delete key=7 (node with only two children) in the example BST, you have to find its immediate successor node (node 8), copy the content (only the key) of its immediate successor node to this node (node 7), and delete the immediate successor node (node 8) from its right subtree.

g. [1 mark] **Overloaded << operator.** The overloaded << operator (friend function) will print out the entire tree level by level. For the example BST, it will print something like this:

10
5 (10->left), 15 (10->right)
3 (5->left), 7 (5->right), 13 (15->left), 18 (15->right)

For illustration purpose, I just show three levels, you should print out the entire tree.

2) [Negative marking] Additional requirements for the *BinarySearchTree* class

a. Make sure the *BinarySearchTree* class is implemented as abstract data type (ADT) in bst.hpp and bst.cpp, respectively.

b. Make sure *BinarySearchTree* class is implemented inside the namespace *ENSC251_Lab3*.

c. For the header filer, use the #ifndef macro to avoid double include.

3) [1.5 marks] Test your simple *BinarySearchTree* in main.cpp:

a. The first part of your main program does not involve user interaction. Do the following. First, create ***BinarySearchTree*** object bst1 and insert the following keys into bst1: {10, 5, 15, 3, 7, 13, 18, 4, 6, 9, 8, 16, 19, 17}. Print out bst1 after inserting each node. Second, create ***BinarySearchTree*** objects bst2 and bst3, and assign bst1 to bst2 to bst3, i.e., bst3 = bst2 = bst1. Print out bst2 and bst3. Third, create ***BinarySearchTree*** object bst4 using copy constructor, i.e., bst4(bst3). Print out bst4. Fourth, delete nodes 4 and 9 from bst1, delete node 10 from bst2, delete nodes 3, 9 and 16 from bst3, delete nodes 18, 7, and 10 from bst4. Print out the corresponding tree after deleting each node. Finally, print out bst1, bst2, bst3, and bst4 again.

b. For the second part of your main program, you will only work on bst4 with user interaction. Print out a menu to the user when it starts running. This menu provides

options for a user to choose among: i) search a key in bst4; ii) search the successor node of a key in bst4; iii) insert a key into bst4; iv) delete a key from bst4; v) print out bst4, and vi) exit the program. Your program should perform accordingly. For option i) and ii), you should print out the returned node, including its key value, the key values of its parent, left and right chidren. For option iii) and iv), you should print out the entire tree of bst4 after the insertion or deletion.

The baseline code for lab assignment 3 can be downloaded from the course website.

In addition to the actual coding implementation, you need to provide good commenting, naming, and other good coding styles; all these count in your lab assignment marking.

Note: For grading logistics and remote machine access, please refer to the course website. If you have any questions, please post them on the discussion board.

Assignment Submission:

Your lab assignment 3 will be submitted electronically through Canvas. You will need to submit a single lab3.zip file. Failure to comply with this format will result in a **ZERO** score for this assignment. To zip your files in Linux,

1. Go to your lab3 directory
2. make clean //**make sure you clean your files**
3. cd .. //go one level up
4. zip -r lab3.zip lab //zip all your lab3 files into a single lab3.zip

Submission Deadline:

Your lab assignment 3 is due at **11:59:59pm on Sunday, Jul 3rd, 2022**. You need to meet the deadline: every 10 minutes late for submission, you lose 1 mark; that is, 100 minutes late, you will get zero for this lab.

Lab Demonstration:

You will have to demo your lab assignment 3 to your TA in the following lab sessions that you enrolled on **Monday (Jul 4th, 2022**). Only code from your Canvas submission is allowed in the lab demo. Each student group has around 10 minutes to explain your code to the TA. If you fail to do the demo (without a medical note), or if it is determined that you do not understand the code being evaluated, you will be awarded zero on this lab assignment. Also please show up in the demo day on time (TA will send out your scheduled time), otherwise you will lose 0.5 mark.