PLEASE NOTE: Before you start remove any flash memory devices from the PC. Failure to unplug flash memory devices may result in an F grade. Leave your phones and bags up at the lecturer machine.
Hand-up: Please include your name as a comment at the top of each .cpp and .h file. Do not Zip. Upload all your .cpp and .h files to Moodle.

Part A (62 Marks)

1. Develop a **Binary Search Tree Node class** that stores integers. The Node only has the following method:

- A node constructor: **TreeNode(int theInt); [4 marks for part A1]**

2. Implement a **Binary Search Tree Class** that stores Tree Nodes. The Tree will have the following methods:

- **BinaryTree()** A tree constructor and creates an empty tree [2 marks]
- **bool isLeaf(TreeNode *)** Returns true if the note is a leaf node [2 marks]
- **int height(Tree Node *)** Note: this a recursive private method which is called by a secure public method. For a tree with just one node, the root node, the height is defined to be 0, if there are 2 levels of nodes the height is 1 and so on. A null tree (no nodes except the null node) is defined to have a height of -1. [4 marks]
- **void add(int)** Note: This should use recursion. Duplicates are not allowed. [4 marks]
- **int delete(int theInt**) Note: deletes the node containing theInt value from the binary search tree if no node has theInt value then return -1. [9 marks]
- **bool find(int theInt)** Returns true if the tree contains the integer value otherwise false. [4 marks]
- **int size()** Returns the number nodes in this binary search tree. [4 marks]
- **void PreOrder(Tree Node *)** Note: this recursive private method is called by a secure public method that outputs a preorder traversal of the tree. [4 marks]
- Provide an overloaded **stream insertion operator <<** as a stand-alone function that outputs all the nodes in the Tree in ascending sequence. [4 marks]
- **int minValue(Tree Node *)** Note: this recursive private method is called by a secure public method that returns the minimum data value found in that tree. [4 marks]
- **~BinaryTree()** Note: A tree destructor that frees up memory [7 marks]

[48 marks in total for part A2]

3. Write a **Main Program** that tests all the above methods of your binary search tree.
[10 marks for part A3]

Please read other side of the page.

Part B (38 Marks)

4. Implement a function **bool isCcomplete(TreeNode* root)** that returns true if the tree at root is a complete BST, false otherwise. [15 marks]

5. Implement a function **bool isBalanced(TreeNode root)** to check if a tree is balanced. For the purposes of this question, a balanced tree is defined to be a tree such that no two leaf nodes differ in distance from the root by more than one. [15 marks]

Further 8 marks for clear well-presented code, well-chosen variable and class names and appropriate comments – if you upload files other than .cpp and .h or fail to put your name on the uploaded files you will lose these marks.

Marks may be deducted for bad programming practice [-50 Marks]
For example, global variables, break, etc, etc.