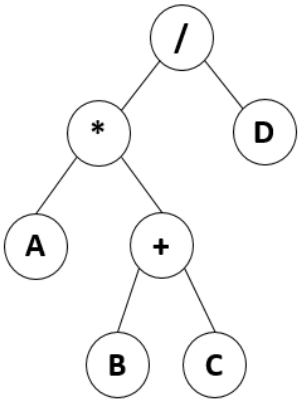
CS 202 Fundamental Structures of Computer Science II
Assignment 2 – Binary Search Trees

Assigned on: 26 October 2016 (Wednesday)
Due Date: 07 November 2016 (Monday) – 23:55

Question-1 (20 Points)

a) (*6 points*) Give the prefix, infix, and postfix expressions obtained by preorder, inorder, and postorder traversals, respectively, for the expression tree below:



b) (*9 points*) Insert 41, 18, 66, 10, 5, 51, 55, 20, 80, 52, 48, 19 to an empty Binary Search Tree, and then delete 10, 55, 66, 18 in given order. Show the evolution of the BST after each insertion and deletion operation.

c) (*5 points*) A binary search tree has a preorder traversal of 15, 2, 8, 6, 4, 10, 9, 12, 18, 16, 25, 20, 22, 21, 23 and an inorder traversal of 2, 4, 6, 8, 9, 10, 12, 15, 16, 18, 20, 21, 22, 23, 25. What is its postorder traversal?

Question-2 – Programming Assignment (60 Points)

You will write a pointer-based C++ implementation of Binary Search Tree with various functions. Each node object will keep an integer data value together with left and right child pointers.

You are required to implement following functions:

bool **isEmpty**(); Tests whether the BST is empty. True if BST is empty; otherwise false.

int **getHeight**(); Gets the height of the BST.

int **getNumberOfNodes**(); Gets the number of nodes in the BST.

bool **add**(const int& newEntry); Adds a new node containing a given data item to the BST. If the data item already exists in the current BST, then don't insert. True if the addition is successful, or false if not.

bool **remove**(const int& anEntry); Removes the node containing the given data item from BST. True if the removal is successful, or false if not.

bool **contains**(const int& anEntry); Tests whether the given data item occurs in the BST. True if the BST contains the given data item, or false if not.

void **preorderTraverse**(); Traverses the BST in preorder and prints data item values in traversal order.

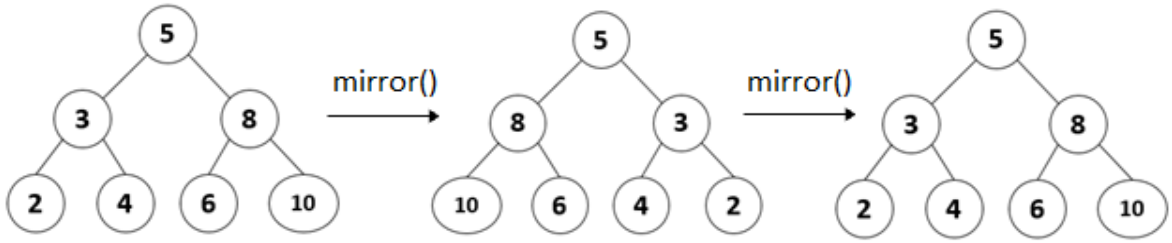
void **inorderTraverse**(); Traverses the BST in inorder and prints data item values in traversal order.

void **postorderTraverse**(); Traverses the BST in postorder and prints data item values in traversal order.

void **levelorderTraverse**(); Traverses the BST in level-order and prints data item values in traversal order. A level-order traversal of a tree processes (visits) nodes one level at a time, from left to right, beginning with the root.

int **span**(const int& a, const int& b); Returns the number of nodes in the BST with data values within a specific range. In other words, it is the number of nodes with the data value >=a and <=b.

void **mirror**(); Changes the BST so that the roles of the left and right pointers are swapped at every node. When you apply this function to a BST, data value in a node will be less than the data values in its left subtree and greater than the data values in its right subtree. If you apply this function to the mirrored tree, you should obtain the original BST.



- Design the node structure together with its set and get methods (and maybe some additional methods if necessary) in **BinaryNode.h** and **BinaryNode.cpp** properly.
- Implement the BST structure with above functions and their interfaces in **BinarySearchTree.h** and **BinarySearchTree.cpp** properly. You are free to write helper functions to accomplish the tasks required from the above functions. **Use the given file names and function signatures during implementation.** You will not submit a main.cpp file, instead we will use our main.cpp file during evaluation. Therefore, it is important to use given file names and function signatures.
- It is recommended to read chapter 15 and 16 in the course textbook to learn the fundamentals and implementation details about Binary Search Trees broadly.

Question-3 – (20 Points)

Explain briefly how you implemented above **levelorderTraverse**, **span** and **mirror** functions and analyze their worst-case running time complexities. **Discuss** whether or not each can be implemented to be asymptotically faster (e.g., $O(n)$ as opposed to $O(n \lg n)$).

IMPORTANT NOTES

Please do not start the assignment before reading these notes.

- Before 23:55 on due date as specified above upload your solutions to Moodle.
- The name of the archive file must be in the following format.
  Surname_Name_StudentId_Section_HW2.zip
- You should upload a single zip file that contains
  - hw2.pdf, the file containing the answers to Q1 and Q3.

    - BinaryNode.h, BinaryNode.cpp, BinarySearchTree.h and BinarySearchTree.cpp, the files containing C++ the source code
    - readme.txt, the file containing anything important on the compilation and execution of your program in Question 2.
    - Do not forget to put your name, student id and section number, in all of these files. Well comment your implementation.
- You are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from somewhere else (e.g., from the internet, other text books, other slides, etc.).
- **IMPORTANT: Although you may use any platform and any operating system in implementing your algorithms and obtaining your experimental results, your code should work in a Linux environment with the g++ compiler. We will test your codes in a Linux environment. Thus, you may lose significant amount of points, if your C++ code does not compile or execute in a Linux environment.**
- Keep all the files until you receive your grade.
- Be careful about indentation.
- Pay attention to these instructions, otherwise you may lose some points even though your code has no error.
- This homework will be graded by your TA, Hasan Balcı (hasan.balci at bilkent edu tr). Thus, you may ask your homework related questions directly to him.

DO THE HOMEWORK YOURSELF. PLAGIARISM AND CHEATING ARE HEAVILY PUNISHED!!!