Ex2

Binary tree

By : Lital Shemsh , Yinon Sinay & Qusai Terabia

This is a two parts project in C++ course in Ariel University . We've been asked to create a binary search tree which capable of basic binary tree attributes(which will be explained later on). We've been made also a test file who checks our tree functionality.

Binary Search Tree is a data structure which gather information in tree like formation. Every tree has a base point which called head or root who made from a destined object called node in general. The node must contains some sort of values and left and right node pointers. The node's right son always has a bigger value and the left son always has a weaker value. In this way every node is naturally sorted.

Our tree is made of struct name "Node" , every node contains two pointers which point at its two son (left or right binary trees), in case the node do not have a one or two sons it will point toward NULL . In addition the node has an integer value.

Functions:

Contains: Boolean function who gets a number from the user and returns whether the tree contains the value or not. It's being done recursively, If the value is bigger than the current node value , the function will be applied on he's right son , if the value is smaller the the function will be applied on the left son. In case the value is equal to the node value than it wil return true. If the function reach to NULL it will return false.

Insert: Function gets an integer from the user and insert it to the tree , if the number already there the program will throw exception. Firstly , we check with Contains if the number in the tree. If not, we will check the head node if its value is higher/lower than the data we inserting. If the head has higher value if will check if it has a left son , if it does we will applied this function on it. Otherwise , we will build a new node with the inserted data. Same goes to the right side if the node has lower number.

Remove: This function gets a number from the user and remove it from the tree. First we use Contains to check if the number exists. If not the program will throw exception , if is does the program will keep trying to search the node. In case the value is higher we

applied Remove recursively on the tree of the right son , some goes to the left son with lower value. Finally when our program reach to the wanted node we check how many son it has. If it has no sons we simply delete it using the destructor. In case the node has one child will replace the node. If the node has two sons the left child will replace the node and the right child will be inserted to the tree as a tree.

InsertTree: This function is being called from Remove. Its goal is to insert a tree to another tree. We check if the value of the head is higher or lower than the value of the other head. If it's higher we check if it has a right son , if not we set its left son as the other tree. In case it has a child we run this function for its left son. And same goes to the right son.

Left: Function that receive integer and returns the node left son. First of all we check if the number exists with Contains , if it not we throw exception. Like Contains we search recursively for the given number. If the number is higher than our current node value we check the right side of the tree , if it lower we check the left side of the tree. Finally when the get to the wanted number we see if it has a left child , if it has we return its value otherwise we throw exception.

Right: Exactly like Left.

Function that receive integer and returns the node right son. First of all we check if the number exists with Contains , if it not we throw exception. Like Contains we search recursively for the given number. If the number is higher than our current node value we check the right side of the tree , if it lower we check the left side of the tree. Finally when the get to the wanted number we see if it has a right child , if it has we return its value otherwise we throw exception.

Parent: This function gets number from the user and return the value of this node parent if it exists. So we check first if the number is in the tree, if it's not we throw exception. Then we search the given number recursively and when it found we return the value.

Root : Returns the head's value if it exists.

Print: Prints the tree nodes in order syle (left child , head , right child).