**Data Structure\_2071035 Lee Somin**

**Technical Report – Tree\_predecessor.cpp**

*Theorical Explanation of Functions in ‘Tree\_predecessor.cpp’*

**typedef struct TreeNode**

the structure contains the data of the node and the left, right nodes’ address, and the address of the parent node.

**tree\_predecessor**

Inputs: TreeNode\* p

Return: TreeNode\* p / TreeNode\* q

This function is made for finding the predecessor of the given tree node \*p. When the node is given, the function determines whether the left subtree of the given node is null. If it is not, the function updates p as the left node once and follows the right link of the tree until there is no node. Then, the function returns the p, which is the updated address of the rightmost node of the left subtree. When the left subtree of the given node is null, the function operates as next. If the parent node of the ‘p’ is null, the target node is the first node. So, there is no predecessor. If the node is right subtree of the parent node, the predecessor of the node is parent node. If the target node has no left subtree and is the left subtree of the parent node, it updates p as q and q as its parent node. When the loop ends, the function returns the predecessor (that has the target node as the right subtree) of the given node, ‘q’.

**main()**

input: non

return: non

In main function, the program assigns the parent nodes of the nodes in the tree. Then, it goes to the rightmost node. Then the program prints the data of the node and updates the previous node(predecessor) of the current node to q using function ‘tree\_predecessor()’ until the predecessor of q is null. Using do-while statement, the loop is executed at least once.

**Result:** 텍스트이(가) 표시된 사진

자동 생성된 설명