PWr Week 7

Algorithms and Data Structures

Laboratory – **List 7**

1. Write a program with below operations for a binary search tree. Elements will be of type `Element`, with fields `key` and `value`.
    a. **`void`** `init(BST& tree)` – which initialize the list `l`.
    b. **`bool`** `insertElem(BST& tree, Element elem)`- insert an element elem in a tree `tree`, depending on field `key` in type `Element`. If there is an element with this same key, do nothing and return `false`. Otherwise – return `true`.
    c. **`void`** `showInorder(BST& tree)` - show elements of tree `tree` using in-order walk. The values are written in one line, after EVERY element (also the last) write a comma ','. Every element have to be written in a format `key`(`value`). If a tree is empty – the line is empty. The line ends with newline character. Example: "1(5),2(2),6(3),"
    d. **`void`** `showPreorder(BST& tree)` - show elements of tree `tree` using pre-order walk. A format like for `showInorder` procedure.
    e. **`void`** `showPostorder(BST& tree)` - show elements of tree `tree` using pre-order walk. A format like for `showInorder` procedure.
    f. **`bool`** `findKey(BST& tree,` **`int`** `key, Element &elem)` - find an element in tree `tree` with value and assign to `elem` this element. Return `true` if element has been found, otherwise – `false`;
    g. **`bool`** `removeKey(BST& tree,` **`int`** `key, Element &elem)` – remove from tree `tree` an element which key is equal to `key` and return it in `elem` parameter. Return `true` if element has been removed, otherwise – `false`;

       If node with the key has two children - in the structure remove the successor (not the predecessor).
    h. **`void`** `clear(BST& tree)` - remove all elements from tree `tree`.
    i. **`int`** `numberOfNodes(BST& tree)` - return number of nodes in a tree `tree`.
    j. **`int`** `height(BST &tree)` – return the height of the tree `tree`.

For **10 points** present solutions for this list till **Week 8**.
For 8 **points** present solutions for this list till **Week 9**.
For 5 **points** present solutions for this list till **Week 10**.
After Week 10 the list is closed.

Advice: many times it is better prepare two functions, one function like in a task and second one with a pointer to a node as a parameter. E.a:

```
void showInorder(Node *node){
     // important code here
}
void showInorder(BST& tree){
     showInorder(tree.root);
     printf("\n");
}
```

Appendix 1

The solution will be automated tested with tests from console of presented below format. The test assumes, that there are up to X different trees, which there are created as the first operation in the test. Each tree can be initialized separately.

If a line is empty or starts from `'#'` sign, the line have to be ignored.
In any other case, your program should print an exclamation mark and write (copy) introduced a line and then, depending on the command follow the correct procedure / function.

If a line has a format:
`GO` *`X`*
your program has to create *n* trees (without initialization). The trees are numbered from 0 like an array of lists. Default current tree is a list with number 0. This operation will be called once as the first command.

If a line has a format:
`CH` *`n`*
your program has to choose a tree of a number `n`, and all next functions will operate on this tree. There is *n>=0* and *n<X*.

If a line has a format:
`IN`
your program has to call `init(t)` for current tree `t`. For any tree this operation will be called once, before using the tree.

If a line has a format:
`IE` *`k v`*
your program has to call `insertElement(t,x)` for current tree `t`, and element x with field `key` equals `k`, and field `value` equals `value`. Write on console returned boolean value.

If a line has a format:
`FK k`
your program has to call `findKey(t,k, el)` for current tree `t`, and if the function return **`true`**, write on the output returned value `el` in format "key(value)". Otherwise write "false" with new line character.

If a line has a format:
`RK` *`k`*
your program has to call `removeKey(t,k, el)` for current tree `t`, and if the function return **`true`**, write on the output returned value `el` in format "key(value)". Otherwise write "false" with new line character.

If a line has a format:
`SI`
your program has to call `showInorder(t)` for current tree `tree`.

If a line has a format:
`SP`
your program has to call `showPreorder(t)` for current tree `tree`.
If a line has a format:
`SQ`
your program has to call `showPostorder(t)` for current tree `tree`.

If a line has a format:
`CL`
your program has to call `clear(t)` for current tree `t`.

If a line has a format:
`NN`
your program has to call `numberOfNodes(t)` for current tree `t` and write in one line returned number.

If a line has a format:
`HE`
your program has to call `height(t)` for current tree `t` and write in one line returned number.

If a line has a format:
`FA`
your program has to call `functionA(t)` for current tree `t` and write in one line returned number.

If a line has a format:
`FB k`
your program has to call `functionB(t,k)` for current tree `t` and write in one line returned number.

If a line has a format:
`FC k`
your program has to call `functionC(t,k, el)` for current tree `t`, and if the function return **`true`**, write on the output returned value `el` in format “key(value)”. Otherwise write “false” with new line character.

If a line has a format:
`HA`
your program has to end the execution, writing as the last line “END OF EXECUTION”. Every test ends with this line.

For example for input test:

```
GO 2
IN
IE 1 4
IE 4 1
IE 3 7
FK 3
```

```
IE 6 10
RK 4
SI
SP
SQ
NN
HE
HA
```

The output have to be:

```
START
!GO 2
!IN
!IE 1 4
true
!IE 4 1
true
!IE 3 7
true
!FK 3
3(7)
!IE 6 10
true
!RK 4
4(1)
!SI
1(4),3(7),6(10),
!SP
1(4),6(10),3(7),
!SQ
3(7),6(10),1(4),
!NN
3
!HE
3
!HA
END OF EXECUTION
```