Exercise #4	Algorithms and Data Structures	
	Topic: Dynamic data structures – trees	Version: 1.0 / 2019
	Prepared by: dr inż. Grzegorz Łukawski & dr inż. Barbara Łukawska	

1) Binary Search Trees (BST)

1.1) The definition

Tree – a recursive dynamic data structure. Tree consists of finite number of nodes which are either empty, or contain disjoint subtrees.

- Its size is unrestricted, it can store initially unknown and variable number of elements (nodes).
- Nodes are linked with pointers, where an element (parent) is linked to more elements (children).
- An element of a tree is a structure, storing a piece of data and an additional pointers to its subtrees.
- Nodes with the same parent are called siblings.

1.2) Types of trees

- **Binary tree** – tree with nodes with at most two children.
- **Binary Search Tree** – tree with ordered nodes: left subtree contains only smaller elements, right subtree only larger.
- **Perfectly balanced binary tree** – a tree, in which the number of nodes in left and right subtree differs by no more than 1.
- **AVL balanced binary tree** – a tree, in which the height of left and right subtree differs by no more than 1.
- **Multiway tree** – a binary tree, where nodes are organized into siblings (right pointer) and children (left pointer).
- **B trees** – trees with values “grouped” into nodes, creating large “pages” for efficient access to memory.
- **B+ trees** – a combination of B-tree and unidirectional list.
- **Tries** – high-order trees storing strings of characters.

1.3) Example implementation of a Binary Search Tree

Data structure:

```
struct Node {
    int transcript;
    float stipend;
    float average;
    struct Node *left;
    struct Node *right;
};
```

Pointer to the first element of the tree (root):

```
struct Node *root = NULL;
```

Inserting a new element into BST, ordered by transcript number (recursive):

```
void insert_item(struct Node **t, int transcript, float s, float a) {
    if (*t == NULL) {
        struct Node *item = new Node;
        item->transcript = transcript;
        item->stipend = s;
        item->average = a;
        // Leaf (no children):
        item->left = NULL;
        item->right = NULL;
        // Modify the pointer:
        *t = item;
    }
    else {
        // Left or right child (may be NULL):
        struct Node *item = *t;
        if (transcript < item->transcript)
                                insert_item(&item->left, transcript, s, a);
        if (transcript > item->transcript)
                                insert_item(&item->right, transcript, s, a);
    }
}
```

Example usage:

```
int transcript, stipend, average;
cin >> transcript >> stipend >> average;
insert_item(&root, transcript, stipend, average);
```

---

2) Exercises

Write a program in **C++**, where the user can add new elements to a BST tree. Sample code shown above may be used for the implementation. Expand your program with the following features:

A) Display content of the tree in ascending and descending order (inorder traversal method);

B) Find the student with given transcript number and display his data;

Additional exercises:

C) Count and display the total amount of money required for stipend for all students;

D) Find and display minimum and maximum transcript number;

E) Find and display minimum and maximum average grade.