Practice Exercise #2 **[Weight: ~3% of the Course Grade]**

Topic: Tree-Based Algorithms and Final Exam Preparation

- **For this assignment, you may work individually or in teams. For individual submission, submit the files to the same team dropbox. Include the name and ID for each team member in your file or files.**
- **Please submit your completed assignment before the dropbox closes on LEARN.**

Step1. Find and Print Sum of Nodes

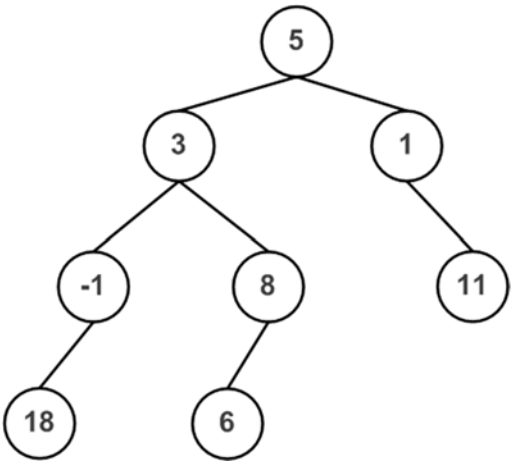
Design and implement a recursive function named **find_and_print_sum_of_nodes** that finds and prints all downwards paths in a given binary tree that sum up to a given value. The paths can start at any node, but do not span both subtrees at that node. The tree is a general binary tree; it is not a heap, BST, or AVL tree.

As input, the function is given: a **BinaryTreeNode* T**, where the function is first called on the **root** node; the desired **sum** value; and a number of other parameters of your choosing. See starting code below.

The function then recursively (1) iterates through the tree levels, (2) keeps track of the discovered paths, and (3) prints each path that sums up to the given **sum** value, including paths that do not start at the root node.

Each **BinaryTreeNode** instance includes as attributes: a **value** of type integer, and pointers to **left** and **right** child nodes. The function has direct access to all members of **BinaryTreeNode**.

For example, for the tree below and sum of 17, your function should output the following: **(5,1,11), (3,8,6), (-1,18)** in any particular order. The order of output paths is not important. Provide three or more test cases (i.e., three different non-empty binary trees) that illustrate that your code runs correctly.




```
void find_and_print_sum_of_nodes (BinaryTreeNode* T,
                                  int desired_sum, int cur_sum, string buffer) {

    // exit if T is NULL
    if (!T) return;

    // update the ongoing sum with the current value for T
    int new_sum = cur_sum + T->value;

    // update the current path string
    string new_buffer = buffer + " " + int2ascii(T->value);

    // if the desired sum is found, print the corresponding path
    if (new_sum == desired_sum)
            cout << new_buffer << endl;

    // TODO: continue down the left subtree

    // TODO: continue down the right subtree

    // TODO: restart from the left subtree if buffer = ""

    // TODO: restart from the right subtree if buffer = ""
```

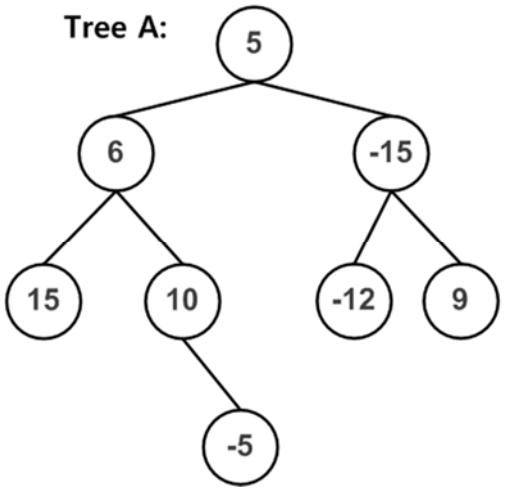
Step2. Find Max Sum of Nodes across Subtrees

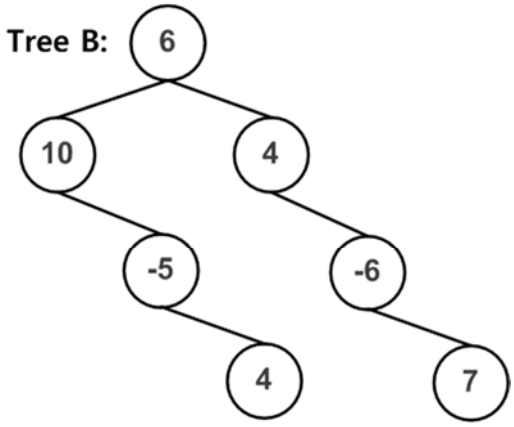
Design and implement a recursive function named **int find_max_sum_of_nodes** that finds a non-cyclical path with the maximum sum in a given binary tree. The paths can start at any node, and can span both subtrees at that node. The tree is a general binary tree that stores integers; it is not a heap, BST, or AVL tree.

As input, the function is given: a **BinaryTreeNode* T**,where the function is first called on the root node, and a number of other parameters of your choosing. The function then recursively (1) iterates through the tree levels, and (2) keeps updating the maximum sum if applicable as new paths are discovered. The function returns the maximum sum back to the caller as an integer value; it does not output the actual maximum path. If the tree is empty, the function returns **0**. See starting code below.

Each **BinaryTreeNode** instance includes as attributes: a **value** of type integer, and pointers to **left** and **right** child nodes. The function has friend access to private members of **BinaryTreeNode**.

For example, for the tree A below, your function should return **31** for the **(15, 6, 10)** path. Similarly, for the tree B below, your function should return **21** for the **(10, 6, 4, -6, 7)** path. Provide three or more test cases (i.e., three different non-empty binary trees) that illustrate that your code runs correctly.






```
int find_max_sum_of_nodes (BinaryTreeNode* T,
                           int &temp_max_sum) {
    // exit if T is NULL
    if (!T) return 0;

    // derive the maximum sum for the left subtree
    int left_sum = find_max_sum_of_nodes(T->left,
                                         temp_max_sum);

    // derive the maximum sum for the right subtree
    int right_sum = find_max_sum_of_nodes(T->right,
                                          temp_max_sum);

    // TODO: compare T->value, left_sum + T->value,
    and right_sum + T->value; store as max1

    // TODO: compare max1, left_sum + right_sum +
                T->value; store as max2

    // TODO: update temp_max_sum with the new max

    // TODO: return max1
}

int find_max_sum_of_nodes(BinaryTreeNode *T) {
  int temp_max_sum = INT_MIN;

  find_max_sum_of_nodes(T, temp_max_sum);

  return temp_max_sum;
}
```