CMPT125, Fall 2019

Homework Assignment 4
Due date: Monday, November 18, 2019, 23:59

You need to implement the functions in ***assignment4.c***.
Submit only the ***assignment4.c*** file to CourSys.

The assignment will be graded both **automatically** and by **reading your code**.

Do not! add main() to your assignemnt4.c file. This will cause compilation errors as the main() function will be in the test file.

Readability: Your code should be readable. Add comments wherever is necessary.
If needed, write helper functions to break the code into small, readable chunks.

Compilation: Your code MUST compile in CSIL with the Makefile provided.
Make sure that your code compiles without warnings/errors,

If the code does not compile in CSIL the grade on the assignment is 0 (zero).
Even if you can't solve a problem, make sure it compiles

Warnings: Warnings during compilation will reduce points.
More importantly, they indicate that something is probably wrong with the code.

Testing: Test your code. An example of a test file is included.
You code will be tested using the provided tests as well as additional tests.
You should create more tests to check your solution.

For the assignment use the following structs for Binary Trees and Binary Search Trees.

```
struct BTnode {
  int value;
  struct BTnode* left;
  struct BTnode* right;
  struct BTnode* parent;
};
typedef struct BTnode BTnode_t;

typedef struct BST {
  BTnode_t* root;
} BST_t;
```

Question 1 [30 points]

Write a function that gets two arrays of length n. The first array is the PreOrder some binary tree and the second array is the InOrder of the binary tree. The functions outputs the binary tree.

```
// the function recovers the tree from its inorder and preorder
BTnode_t* reconstruct_tree(int* preorder, int* inorder, int n)
```

Question 2 [10 points]

Write a function that gets a binary tree and returns the sum of its elements.

```
// returns the sum of elements in the tree
int get_sum(BTnode_t* tree)
```

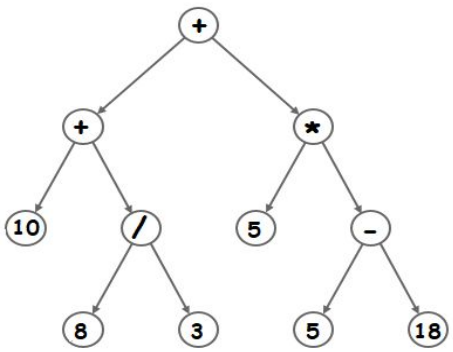
Question 3 [25 points]

Write a function that gets a binary tree representing an arithmetic expression and returns a string containing the expression. For operations we use the following enum.

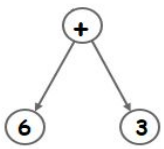
```
enum {PLUS, MINUS, MULT, DIV};

// converts a arithmetic expression from tree to string
char* get_arithmetic_expression(BTnode_t* expression)
```

The expression must have parentheses for each operation (including the outermost parentheses), and all tokens (numbers and operations) must be separated by single spaces. See more examples for the exact format in the test file.



((10 + (8 / 3)) + (5 * (5 - 18)))



(6 + 3)

8

Question 4 [10 points]

Write a function that gets a root of a Binary Search Tree and finds the smallest element in the tree.

```
// returns the minimal element in the tree
// assumption: tree is not empty
int get_min(BST_t* tree)
```

Question 5 [25 points]

Write a function that gets a Binary Search Tree and a number k, and returns the sum of all elements in the tree that are greater than k.

```
// returns the sum of elements in the tree greater than k
int sum_greater_than_k(BST_t* tree, int k)
```

Implement the function as efficiently as possible. For example, if the root is less than k, you should not explore the left subtree.
The runtime of the function should depend on the number of elements that are greater than k, and not necessarily on the size of the entire tree.