

CMPT125, Fall 2025

Homework Assignment 5

Due date: Sunday, November 30, 2025, 23:59

You need to implement the functions in ***assignment5.c***.
Submit only the ***assignment5.c*** file to CourSys.

Solve all problems in this assignment, one function for each problem.

Grading: The assignment will be graded automatically.

Make sure that your code compiles without warnings/errors, and returns the required output.

Compilation: Your code MUST compile in CSIL with the Makefile provided.

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 the file compiles properly.

Warnings: Warnings during compilation will reduce points.

More importantly, they indicate that something is probably wrong with the code.

Dynamically allocated arrays: Do not use variable length arrays! Never!

If you need an array of unknown length, you need to use malloc.

Memory leaks: Memory leaks during execution of your code will reduce points.

Make sure all memory used for intermediate calculations are freed properly.

Readability: Your code must be readable, and have reasonable documentation, but not too much. No need to explain `i+=2` with `// increase i by 2`.

Write helper functions if that makes the code more readable.

Testing: An example of a test file is included.

Your code will be tested using the provided tests as well as additional tests.

Do not hard-code any results produced by the functions as we will have additional tests.

You are strongly encouraged to write more tests to check your solution is correct, but you don't have to submit them.

1. You need to implement all the functions in ***assignment5.c***.
2. You should not add `main()` to `assignment5.c`, because it will interfere with `main()` in the test file.
3. Submit only the ***assignment5.c*** file to CourSys.

Write the following operations on a Binary Search Tree:

- 1) [20 pts] The function gets the root of a Binary Tree, and checks if it is a Binary Search Tree.

```
bool is_BST(BTnode_t* root);
```

- 2) [20 pts] The function gets a Binary Search Tree, and returns the struct min_max containing the minimal and the maximal element in the tree.

If the tree is empty, the function returns 0 in both fields.

See assignment5.h for the definition of the struct min_max

```
min_max BST_min_max(BST_t* bst);
```

- 3) [20 pts] The function gets a Binary Search Tree, and returns an array that contains all values from the tree sorted in the non-decreasing order.

The length of the array will be equal to the size of the tree.

Assumption: the tree is not empty

```
int* BST_to_array(BST_t* bst);
```

- 4) [20 pts] The function gets an array of length n of ints sorted in the non-decreasing order, and returns a Binary Search Tree containing all the values in the array, such that the depth of the tree is at most $\log_2(n) + 1$.

```
BST_t* sorted_array_to_BST(int* ar, int n);
```

- 5) [20 pts] The function gets an array containing the pre order traversal of a Binary Search Tree, and returns the Binary Search Tree.

```
BST_t* preorder_to_BST(int* preorder, int n);
```