

CMPT125, Fall 2025

Homework Assignment 4

Due date: Sunday, November 16, 2025, 23:59

You need to implement the functions in **assignment4.c**.
Submit only the **assignment4.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 **assignment4.c**.
2. You should not add `main()` to `assignment4.c`, because it will interfere with `main()` in the test file.
3. Submit only the **assignment4.c** file to CourSys.

Problem 1 [30 points]

- When writing your solution for this problem, **you may only use standard functions of `queue`** (`queue_create`, `enqueue`, `dequeue`, `queue_is_empty`, `queue_free`).
- The signatures of the functions are given in `queue.h`.
- **You should NOT make any assumptions about how `queue_t` is implemented.**
- The provided implementation of `queue_t` is obfuscated on purpose, and it might change between test cases.

Write the following functions on a queue of ints.

You may assume the provided arguments to the functions are not NULL.

a) [10 pts] The function gets a queue, and returns the number of elements in the queue.

```
// gets a queue of chars and returns the number of elements in queue
// when the function returns, q should be in the same state
// as it was in the beginning
int queue_size(queue_t* q);
```

b) [20 pts] The function gets two queues, and checks if the queues have the same state, i.e., if they have exactly the same elements in them in the same order.

```
// checks if the two queues are equal
// when the function returns, q1 and q2 should be in the same state
// as they were in the beginning
bool queue_equal(queue_t* q1, queue_t* q2);
```

Problem 2 [30 points]

Write the following operations on a binary tree:

a) [15 pts] The function gets a root of a Binary Tree, and a boolean predicate `pred`. It returns a pointer to a node whose value satisfies `pred(node->value)==true`. If no such node is not found, the function returns NULL. If there are several such nodes, the function may return any of them.

```
BTnode_t* find(BTnode_t* root, bool (*pred)(int));
```

b) [15 pts] The function gets a root of a Binary Tree of ints, and a function `f`. It applies `f` to the value of each node in the tree.

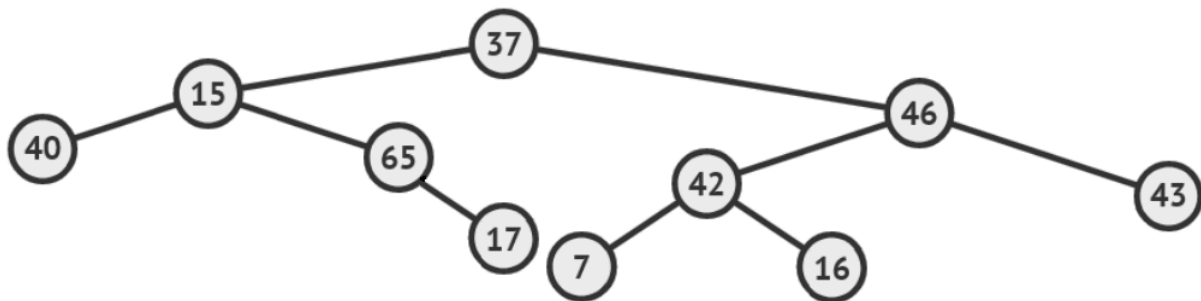
```
void map(BTnode_t* root, int (*f)(int));
```

Problem 3 [40 points]

Write a function that gets a node in a binary tree, and returns the next node in the inorder traversal of the tree.

```
BTnode_t* next_inorder(BTnode_t* node)
```

For example, consider the following tree.



The inorder traversal of this tree is [40, 15, 65, 17, 37, 7, 42, 16, 46, 43]

- On input 40 the output needs to be the node containing 15.
- On input 15 the output needs to be the node containing 65.
- On input 65 the output needs to be the node containing 17.
- On input 17 the output needs to be the node containing 37.
- And so on...
- On input 43 the output needs to be NULL.

A typical test case will start with the first node of the inorder traversal, and traverse the entire tree.

```
BTnode_t* node = get_root();  
while (node) {  
    node = next_preorder(node);  
    // check that the output is correct  
}
```

For full marks your solution needs to traverse a tree of size up to 100,000 under one second.

*** You may assume the input is always a node in a tree with all pointers assigned properly.**