CMPT125, Spring 2022

Homework Assignment 4
Due date: Friday, March 25, 2022, 23:59

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

Solve all 3 problems in the assignment.

The assignment will be graded automatically.
Make sure that your code compiles without warnings/errors, and returns the required output.

* For Question 1 the assignment contains the file ***queue.o*** *implementing queue of chars.*
** Y**ou may only use the signatures*** of the functions iven in queue.h.
**** You do not know how exactly queue_t is implemented****.*
** Furthermore, the exact implementation details might change between test cases.*
** The provided .o file* should *work on the CSIL environment, but probably not on Windows.*
** You can compile your own .o file by running "gcc -c queue.c"*

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.

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

Memory leak during execution of your code will reduce points.
Check that all memory used for intermediate calculations are freed properly.

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

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 the functions in ***assignment4.c****.*
2. If necessary, you may add helper functions to the assignment4.c file.
3. You should not add main() to assignment4.c, because it will interfere with main() in the test file.
4. Submit only the ***assignment4.c*** file to CourSys.

Question 1 [40 points]

Write the following functions on a queue of chars.
When writing the functions, you ***may only use the signatures*** *given in queue.h,*
but ***you don't know how exactly queue_t is implemented****.*
Furthermore, the exact implementation details might change between test cases.

You may assume the provided queues are not NULL.

a) [10 points] The function gets two queues, dest and source, and moves all nodes from source to dest.
It returns the number of moved nodes.
For example, if source=[a,b,c,d,e] and dest=[x,y,z], then after applying the function we'll have source=[] and dest=[x,y,z,a,b,c,d,e], and the function returns 5.
[Assuming we add to the queue from the right, and remove from the left]

```
// moves all elements from source to destination in the FIFO order
// returns the number of elements moved
int queue_move(queue_t* dest, queue_t* source);
```

b) [10 points] 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);
```

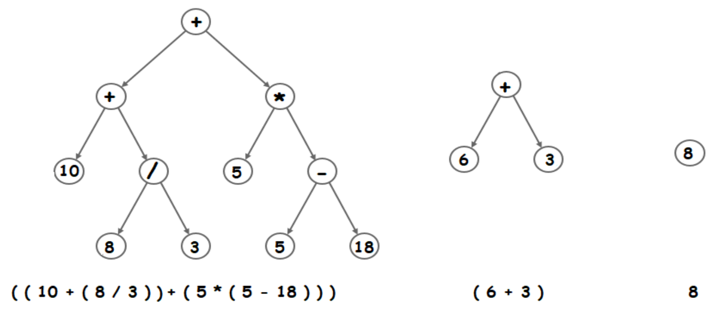
c) [20 points] The function gets two queues, and checks if the queues have the same state, i.e., if they have exactly the same elements in them in the same order.

```
// 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* q1);
```

Question 2 [20 points]

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

```
enum {PLUS = '+', MINUS = '-', MULT = '*', DIV = '/'};
// evaluates an arithmetic expression in the given tree
// assumption: tree is not null and not empty
// assume all internal nodes are operations as in the enum.
float eval_arithmetic_expression(BTnode_t* root)
```



The outputs for the trees are 10+2.6666+ 5*(-13)=-52.33333, 6+3 = 9, and 8.
(Here you may find the switch-statement useful.
https://www.programiz.com/c-programming/c-switch-case-statement)

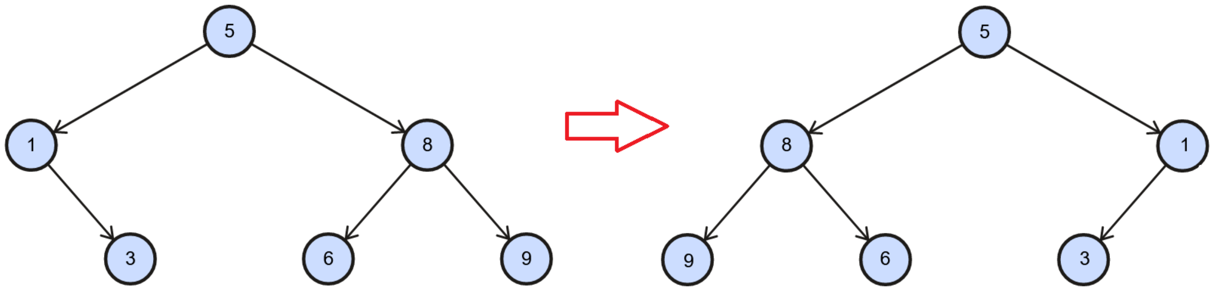
Question 3 [40 points: 20 points each]

Write the following functions on binary trees.

a) The function gets a binary tree, and a predicate pred, and returns the pointer to the node satisfying pred. If there is more than one such node, the function returns the points to the first such node according to the in-order traversal.

```
// returns the first node satisfying pred according to in-order
traversal
// if such node not found, returns NULL
BTnode_t* find(BTnode_t* root, bool (*pred)(int));
```

b) The function gets a binary tree, and creates a new binary tree with a mirror structure (holding the same values inside). See example below.



*** For each node in the original tree, you need to create a new node that will store the copy in the mirror tree.*

```
// creates a new tree that is the mirror image of the input
// the nodes in the new tree should all be new
// and should not use the nodes of the original tree
// if root == NULL, the function returns NULL
BTnode_t* create_mirror_tree(BTnode_t* root);
```