CMPT125, Fall 2021

Homework Assignment 4
Due date: Friday, November 18, 2022, 23:59

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

Solve all 3 problems in this assignment.

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.
You will need to create an object file *queue.o*.
This is done using the command "gcc -c queue.c" in the lib folder

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.
1. Submit only the ***assignment4.c*** file to CourSys.

Problem 1 [40 points]
In this question we get a queue of strings. The implementation is in a separate file. You should not make assumptions about the exact implementation details. You may only use the following functions to access the queue.

```c
typedef struct {
 // not known
} queue_str_t;
// creates a new queue
queue_str_t* queue_create();
// enqueue a given item to the queue
void enqueue(queue_str_t* q, const char* str);
// dequeue the next element from the queue and returns it
// Pre condition: queue is not empty
char* dequeue(queue_str_t* q);
// checks if the queue is empty
bool queue_is_empty(queue_str_t* q);
// free the queue
void queue_free(queue_str_t* q);
```

a) *[10 pts] Write a function that gets a queue of strings and returns the number of elements in it. When the function returns, the queue must be in its initial state.*
```c
// returns the size of the queue
int queue_size(queue_str_t* q)
```

b) *[15 pts] Write a function that gets two queues of chars and checks if they are equal, i.e., have equal strings in the same order. You need to use strcmp() to check if the strings are equal. When the function returns, the queues must be in their initial state.*
```c
// checks if the two queues are equal
bool queue_equal(queue_str_t* q1, queue_str_t* q2)
```

c) *[15 pts] Write a function that gets a queue of strings and returns the string consisting of the concatenation of all the strings in it. When the function returns, the queue must be in its initial state.*
```c
// returns the concatenation of all strings in the queue
// For example, if we add to the queue "a ", then "_b" and
then "@ C2D", the function returns the string "a _b@ C2D".
char* queue_str_to_string(queue_str_t* q)
```

- ***Remember: you should only use the provided interface, and not assume that queue_str_t is implemented in a certain way. For grading your solution the implementation might (and will) change.***
- **Use the object file *queue.o* that contains the implementation of the queue.**
- **You can build your own .o file using "*gcc -c queue.c*" in the lib folder.**

For Problems 2-3 use the following struct representing a node in a Binary Tree.

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

Problem 2 [30 points]

Write the following operations on a binary tree:

a) *[10 pts] The function gets a root of a Binary Tree, and a boolean predicate pred.*
It returns a pointer to a node v whose data satisfies pred(v->data)==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.

```
// finds the a node satisfying pred
// if such node not found, 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) *[10 pts] The function gets a root of a Binary Tree of ints, and a function f.*
It applies f to the data of each node in the tree.

```
// appies f to each node of the tree
void map(BTnode_t* root, int (*f)(int));
```

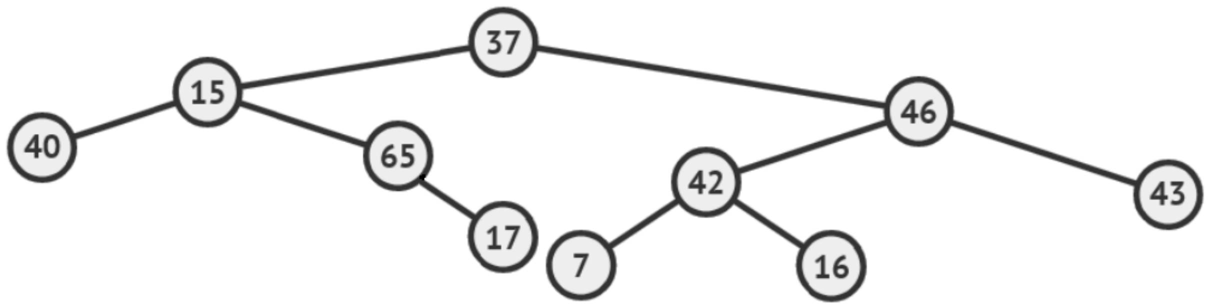
c) *[10 pts] The function gets a root of a Binary Tree of ints, and computes the sum of all leaves in the tree.*

```
// computes the sum of all leaves of the tree
int sum_of_leaves(const BTnode_t* root);
```

Problem 3 [30 points]
Write a function that gets a node in a binary tree, and returns the next node in the preorder traversal of the tree.

```
BTnode_t* next_preorder(BTnode_t* node)
```

For example, consider the following tree.



The preorder traversal of this tree is [37, 15, 40, 65, 17, 46, 42, 7, 16, 43]
- *On input 37 the output needs to be the node containing 15.*
- *On input 15 the output needs to be the node containing 40.*
- *On input 40 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 46.*
- *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.**