CMPT125, Fall 2021

Homework Assignment 4
Due date: Wednesday, November 19, 2021, 23:59

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

Solve all 3 problems in the assignment.

The assignment will be graded both **automatically** and by **reading your code**.
.
Correctness: Make sure that your code compiles without warnings/errors,
and returns the required output.

Readability: Your code should be readable. Add comments wherever 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.
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.

Helper functions: If necessary, you may add helper functions to the .c file.

main() function: do not add main(). Adding main() will cause compilation errors, as the main()
function is already in the test file.

Using printf()/scanf(): Your function should have no unnecessary printf() statements. They
may interfere with the automatic graders.

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

Testing: An example of a test file is included.
Your code will be tested using the provided tests as well as additional tests.
You are *strongly encouraged to write more tests* to check your solution is correct, but you don't
need to submit them.

You need to implement the functions in *assignment4.c*.
If necessary, you may add helper functions to the assignmen4.c file,
but you should not add main() to the file.
Submit only the *assignment4.c* file to Canvas.

Question 1 [40 points]

In this question we get a stack of chars. The implementation is given in a separate file.
You should not make assumptions about the exact implementation details.
You may only use the following functions to access the stack.

```
typedef struct {
  // not known
} stack_t;

// creates a new stack
stack_t* stack_create();
// pushes a given item to the stack
void stack_push(stack_t* s, char item);
// removes the top element from the stack and returns it
// Pre condition: stack is not empty
char stack_pop(stack_t* s);
// checks if the stack is empty
bool stack_is_empty(stack_t* s);
// frees the stack
void stack_free(stack_t* s);
```

a) *[10 pts] Write a function that gets a stack of chars and returns the number of elements in it. When the function returns, the stacks must be in their initial state.*

```
// returns the size of the stack
int stack_size(stack_t* s)
```

b) *[15 pts] Write a function that gets two stacks of chars and checks if they are equal (i.e., have the same elements in the same order). When the function returns, the stacks must be in their initial state.*

```
// checks if the two stacks are equal
bool stack_equal(stack_t* s1, stack_t* s2)
```

c) *[15 pts] Write a function that gets a stack of chars and returns the string consisting of the chars in it. When the function returns, the stack must be in its initial state.*

```
// converts stack to a string
// For example, suppose we push 'A', then 'B', and then
// 'C'. The function needs to return the string "ABC".
char* stack_to_string(stack_t* s)
```

** Remember: you should only use the provided interface, and not assume that stack_t is implemented in a certain way. For grading your solution this

implementation will change.

**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;
```

Question 2 [30 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(const BTnode_t* expression)
```

The expression must have parentheses for each operation (<u>except for the outermost parentheses</u>), and all tokens (numbers and operations) must be separated by a single space. See more examples for the exact format in the test file.
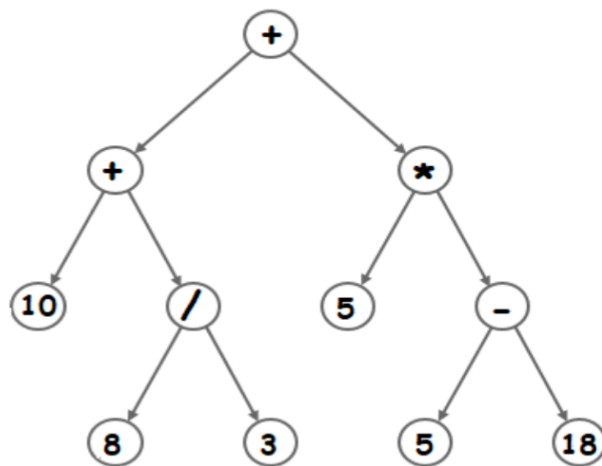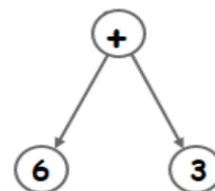
(10 + (8 / 3)) + (5 * (5 - 18))   6 + 3   8

** Negative values are allowed,
** You may assume that all numbers are at most 3 digits long (incl. minus sign)
** YOu may assume the expression is alway legal
** You may find the function *sprintf()* useful here. The function is similar to printf(), but prints to string, e.g, *sprintf(str, "%d", 15);* prints 15 to str.

Question 3 [30 points]

Write the following functions:

a) *[10 pts] The function gets a root of a Binary Tree of ints, and a boolean predicate pred. It returns a pointer to a vertex v for which the data satisfies pred, i.e. pred(v->>data)==true.*
   If no such vertex is not found, the function returns NULL.
   If there are several such vertices, the function returns the first such v in pre-order .

```
// finds the first node in pre-order satisfying pred
// if such node not found, returns NULL
BTnode_t* find(const 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 (including the root)
void map(BTnode_t* root, int (*f)(int));
```

c) *[10 pts] The function gets a root of a Binary Tree of ints, and creates another copy with exactly the same structure, and the same values inside.*
   ** For each node in the original tree, you need to create a new node that will store the copy

```
// creates a new copy of the tree
BTnode_t* copy_tree(const BTnode_t* root);
```