CMPT125, Fall 2020

Homework Assignment 4
Due date: Wednesday, November 25, 2020, 23:59

For this assignment you need to:

1) Implement your solutions to problems 1 and 2 in assignment4.c
2) Implement your solutions to problem 3 in my_array.h and my_array.c

Submit all files to CourSys.

The sum of all points for the 3 problems is 120.

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 is 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 assignment1.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(): 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: Examples of tests are included in *test_trees.c* and *test_my_array.c*.
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.
When working on one problem, comment out all other tests.

Good luck!

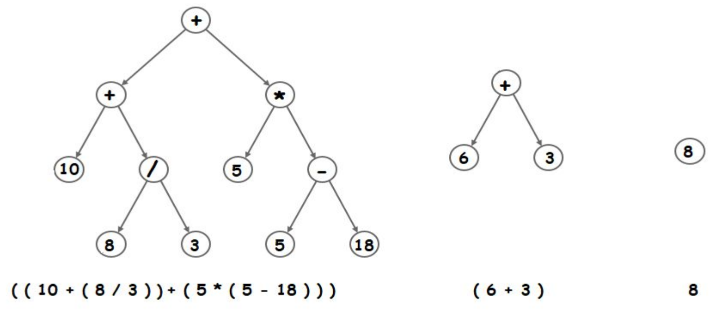
Implement your solutions to problems 1 and 2 in assignment4.c
For Problems 1-2 use the following structs for Binary Tree and Binary Search Tree.

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

typedef struct BST {
  BTnode t* root;
} BST_t;
```

Problem 1 [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.

```
typedef enum {PLUS, MINUS, MULT, DIV} bin op;
// evaluates an arithmetic expression in the given tree
// assumption: tree is not null and not empty
// assume all internal nodes are bin ops, i.e. between 0 and 3
float eval_arithmetic_expression(BTnode_t* root)
```



The outputs for the tree above should be 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)

Problem 2 [15 points each]
Implement the following two functions on Binary Search Trees.
In both questions you may assume the tree in not null and not empty

```
// returns the maximal element in the tree
int get_max(BST_t* tree)

// returns the median element in the tree. For example,
// if the tree contains {1,3,6,8,100}, the output should be 6
// if the tree contains {1,2,3,7,8,100}, the output should be 7.
int get_median(BST_t* tree)
```

Problem 3 [70 points]
Implement your solutions to problem 3 in my_array.h and my_array.c

In this problem you need to implement the ADT my_array of ints.
You need to implement the data structure to support the functions stated below.
Try to implement all functions as efficiently as possible.

For all questions below in all three parts you may assume that the arguments are legal (e.g., my_array is not NULL, length>0, index in the array is between 0 and length-1, etc)

Part 1 [20 points]

```c
// The struct represents an array with indices 0..length-1
// The length of the array can be set in create_my_array().
// The length can be obtained using my_ar_get_length.
// The length can be modified using my_ar_resize.
typedef struct {
	// implement me
} my_array;

// creates a new my_array of the given length
// the entries of the array are not initialized
// if malloc fails, returns NULL
my_array* create_my_array(int length);

// Returns the length of the array.
// Assumption: ar has been created correctly.
int my_ar_get_length(const my_array* ar);

// Sets the value in the array at the specified index.
// Returns the item that was added.
int my_ar_set_value(my_array* ar, int index, int item);

// Returns the value from the array at the specified index.
int my_ar_get_value(const my_array* ar, int index);

// Frees the array
void my_ar_free(my_array* ar);
```

Part 2 [20 points]

```c
[6 points]
// Changes the size of the array.
// If new_size is greater than the original length
// it copies the original content into the new array
// and leaves the remaining entries uninitialized.
// If new_size is smaller than the original length
// it copies the first new_size elements into the new array.
// Returns new_size if all is ok. If memory allocation fails returns -1
int my_ar_resize(my_array* ar, int new_size);
```

```
[6 points]
// Creates a copy of the given my_array and returns it.
// Returns NULL if memory allocation fails.
my_array* my_ar_copy(const my_array* src);


[8 points]
// Appends src to the end of dest.
// Returns the pointer to the resulting dest.
// Returns NULL if memory allocation fails.
my_array* my_ar_append(my_array* dest, const my_array* src);
```

Part 3 [30 points]

```
[6 points]
// Returns the first index of the array containing the element.
// If element not found, returns -1.
int my_ar_find(const my_array* ar, int element);

[6 points]
// For each index i=0..length-1 applies the function f to ar[i]
// and stores f(ar[i]) in ar[i].
void my_ar_map(my_array* ar, int (*f)(int));

[8 points]
// Start with accumulator = ar[0]
// For i=1...length-1 compute accumulator=f(accumulator, ar[i])
// Return accumulator.
// For example, if f computes the sum of two input,
// then reduce will compute the sum of the entire array.
int my_ar_reduce(const my_array* ar, int (*f)(int,int));


[10 points]
// Returns a new my_array containing only the elements of src
// satisfying f(element)==true.
// The length of the new array is adjusted accordingly.
// The order of the elements in the resulting array might change!
my_array* my_ar_filter(my_array* src, bool (*f)(int));
```