# ESC190 LAB 2

The Last Library

**Due**: February 23, 2020 23:59

## INSTRUCTIONS

1. `git pull` to access the provided starter code. You will find the code under `<utorid>/lab2/`
2. **Only** the file `lab2.c` will be marked. You may modify `main.c`, `test.c` and `test.h` to test your code.
3. Your code must compile and run on the ECF with GCC version 8.3.1.

## BACKGROUND

The year is 2060. The world has been taken over by a totalitarian government that has banned all books from publication as part of their propaganda agenda. You are a part of the Resistance, a secret underground society that aims to restore the world to its previous state. You find yourself in charge of "The Library", a network of secret locations around the globe housing clandestine copies of what remains of the world's literature. Unfortunately, the world's government has become aware of the Library, and has been putting all its might towards shutting it down. A number of cyber-attacks launched against the Resistance's computers has scrambled the ordering of the book IDs within the Library's computer system, making it nearly impossible for the people working at the Library to search for books or keep tabs on which books are where. As the head of the Library, it's your responsibility to thwart the attacks. Your immediate task is to get the book IDs back in their sorted state as soon as possible. You decide to tackle this issue with your knowledge of efficient sorting using **heaps**. You are also familiar with the arcane, but highly memory efficient programming language C, which you intend to leverage to get the Library back up and running as soon as possible.

## PROVIDED CODE

- `lab2.c`: This is the **only file that will be graded**.
- `lab2.h`: Contains the declarations for the functions that you are to implement in `lab2.c`. **Do not modify this file.**
- `main.c`: This is the main file for the program. You may modify it to test your code. You'll see that by default, `main()` calls the testing functions that we have provided for you. Your modifications to this file will not be marked.
- `test.c`: Contains functions that allow you to test your code against simple test cases. This test suite **is not complete,** and you should test your code thoroughly with additional test cases which you may add to this file. Your modifications to this file will not be marked.
- `test.h`: Contains the declarations for the test functions. Modify this file if you add more functions to `test.c`. Your modifications to this file will not be marked.

Note that when compiling (see below), all of the above files will be linked together into a single program.

## COMPILING AND RUNNING YOUR CODE

You can generate the executable for your code by compiling and linking using the following command:

```
gcc main.c lab2.c test.c -o lab2
```

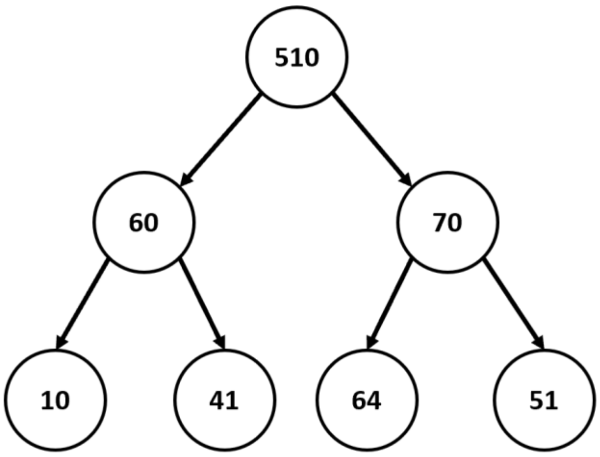
And run your code with the following:

```
./lab2
```

Note that **all your files** must compile without syntax errors to be able to update the executable.

## PRELAB

Convert the following heap into array form:



| **Value** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Index** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |

Indicate the formula to find:

The left child index from parent index $i$: ________________________

The right child index from parent index $i$: ________________________

The parent index from child index $j$: ________________________

## TASKS

Complete the following functions. **These functions can be completed independently**. Note that **none of the functions require you to create new arrays** – for the functions that require you to reorder array elements (namely `heapify()` and `heapsort()`), you must do so **in-place**.

### `void print_tree(float arr[], int n);`

Print the elements of the array (with length n) such that element 0 is on the first line, elements 1 and 2 are on the next line, and so on. The maximum number of elements on the $i^{th}$line is $2^{i-1}$. The last line printed should have 4 spaces between each value. All other lines should be formatted such that the parent value's first digit is equidistant to each child value. If there is an even number of spaces between the parent's two children, the parent's first digit must be one space closer to the left child.

**Input format**

`float[] arr`: an array

`int n`: the number of elements in `arr`

**Sample inputs and outputs**

```
float arr[7] = {40, 20, 17, 16, 13, 15, 11};

print_tree(arr, 7);

Stdout:\n            40
               20          17
            16    13    15    11
```

### `float get_parent_value(float arr[], int n, int index);`

Given a heapified array of length n and an index, return the value of the parent node of the node at `index`. If `index` is invalid, i.e. if it is out of the array's bounds or equal to 0 (the root node), return -1.

**Input format**

`float[] arr`: a heapified array

`int n`: the number of elements in `arr`

`index`: the index of the node whose parent's value we want to find

**Sample inputs and outputs**

```
float arr[3] = {15, 8, 13};

float parent = get_parent_value(arr, 3, 1);

printf("%f", parent);

Stdout:\n 15
```

## `float get_left_value(float arr[], int n, int index);`

Given a heapified array of length n and an index, return the value of the left child node of the node at `index`. If `index` is invalid, i.e. if it is out of the array's bounds or refers to a node without a left child, return -1.

**Input format**

`float[] arr`: a heapified array

`int n`: the number of elements in `arr`

`index`: the index of the node whose left child value we want to find

**Sample inputs and outputs**

```
float arr[3] = {15, 8, 13};

float left_child = get_left_value(arr, 3, 0);

printf("%f\n", left_child);

Stdout:\n 8
```

## `float get_right_value(float arr[], int n, int index);`

Given a heapified array of length n and an index, return the value of the right child node of the node at `index`. If `index` is invalid, i.e. if it is out of the array's bounds or refers to a node without a right child, return -1.

**Input format**

`float[] arr`: a heapified array

`int n`: the number of elements in `arr`

`index`: the index of the node whose right child value we want to find

**Sample inputs and outputs**

```
float arr[3] = {15, 8, 13};

float right_child = get_left_value(arr, 3, 0);

printf("%f\n", right_child);

Stdout:\n 13
```

## `int is_max_heap(float arr[], int n);`

Return 1 if arr is a max heap. Return 0 otherwise.

**Input format**

`float[] arr`: an array that we are testing for the heap property

`int n`: the number of elements in `arr`

**Sample inputs and outputs**

```
float arr1[3] = {8, 15, 13};

float arr2[3] = {15, 8, 13};

int heap_status1 = is_max_heap(arr1, 3);

int heap_status2 = is_max_heap(arr2, 3);

printf("Arr 1 got %d. Arr 2 got %d.\n", heap_status1, heap_status2);

Stdout:\n Arr 1 got 0. Arr 2 got 1.
```

## `void heapify(float arr[], int n);`

Given an array of length n, reorder its elements **in-place** such that the array satisfies the max heap property. You may use the method you saw in lecture.

**Input format**

`float[] arr`: an array that we want to heapify

`int n`: the number of elements in `arr`

**Sample inputs and outputs**

```
float arr[3] = {8, 15, 13};

heapify(arr, 3);

printf("The array is {%f, %f, %f}.\n", arr[0], arr[1], arr[2]);

Stdout:\n The array is {15, 8, 13}.
```

### `void heapsort(float arr[], int n);`

Given a heapified array (max heap) of length n, reorder its elements **in-place** such that the array is now sorted.

**Input format**

`float[] arr`: a heapified array that we want to sort

`int n`: the number of elements in `arr`

**Sample inputs and outputs**

```
float arr[3] = {15, 8, 13};
heapsort(arr, 3);
printf("The array is {%f, %f, %f}.\n", arr[0], arr[1], arr[2]);
Stdout:\n The array is {8, 13, 15}.
```

### `float find_most_common_element(float arr[], int n);`

Given an array of length n, find its most common element.

**Input format**

`float[] arr`: an array whose most common element we want to find

`int n`: the number of elements in `arr`

**Sample inputs and outputs**

```
float arr[6] = {7, 3, 10, 7, 3, 9};
float el = find_most_common_element(arr, 6);
printf("The most common element is %f.\n", el);
Stdout:\n The most common element is 7
```

OR

```
Stdout:\n The most common element is 3
```

## Submission Instructions

To submit: stage, commit, and push your code. Don't forget that `lab2.c` is the only file that will be marked. See the ESC190 Handbook for a refresher on Git instructions.