CSCE 240 HOMEWORK 3 DUE: 05 OCT 2018

You shall submit a zipped, **and only zipped**, archive of your homework directory, hw3.
The directory shall contain, at a minimum, the files `max_heap.h` and `max_heap.cc`.
I will use my own makefile and tests to evaluate your library.

"Hurricane Florence" has left us a week behind. We should be working on a project involving objects, but have not covered that material. This leaves us having one more function library assignment. This is not a useless task. If you program in languages like JavaScript or Python(especially using the Django framework), you may find yourself building modules of functions similar to the library we are creating in this assignment.

There is a well-studied data structure called a binary heap. It is very efficient in maintaining and providing a list of minimum or maximum values from a set. It is nearly always used in implementing a priority queue—a scheduling tool. Your understanding of this structure will be expected in CSCE 311. They are used in managing processes in a CPU.

I would like to be able to treat an integer array as a max heap.

A max heap uses a binary tree stored in a special way. Rather than nodes and pointers, the entire structure is kept in an array. The left and right children of each node at index n are stored at indices $2n + 1$ and $2n + 2$, respectively. I would highly recommend making making two functions,
`inline int left(int root);` and
`inline int right(int root);`.
Each function simply returns the calculation, where root is n. Less mess, easier to read, and little chance of calculation errors. One more function which will be **highly** valuable is
`inline int parent(int child);`.
This function inspects the child's index, determines if it is a right or left child and uses that information to calculate the parent index of the given child index.

You must provide the following functions:

- `void Add(int item, int heap[], int count)`
  - Function accepts an item to be added to heap, an array representing the heap, and count, the current number of items in the heap.
  - Precondition(s):
    - Size of int array heap is greater than count
  - Postcondition(s):
    - item is stored in heap
    - Heap Property: For each node in the tree stored in heap, a node's children's values are less than the node's value.
  - Algorithm:
    1. Add the element to the bottom level of the heap.
    2. Compare the added element with its parent; if they are in the correct order, stop.
    3. If not, swap the element with its parent and return to the previous step.

- `int Remove(int heap[], int count)`
  - Accepts heap, an int array representing the heap and count, the current number of items in the heap. Function removes the largest value from heap and returns that value.
  - Precondition(s):
    - Size of int array heap is greater than or equal to count
  - Postcondition(s):
    - Largest value removed from heap and returned by function
    - Heap Property: For each node in the tree stored in heap, a node's children's values are less than the node's value
  - Algorithm:
    1. Replace the root of the heap with the last element on the last level.
    2. Compare the new root with its children; if they are in the correct order, stop.
    3. If not, swap the element with its LARGER child and return to the previous step.
- `void MaxHeapify(int array[], int count):`
  - The function accepts an integer array array, and the size of the array count, and performs an **IN-PLACE** conversion to a binary max heap, likely using the function Add.
  - Precondition(s):
    - Size of int array array is greater than or equal to count
  - Postcondition(s):
    - Heap Property: For each node in the tree stored in array, a node's children's values are less than the node's value
- `void HeapSortAsc(int array[], int count):`
  - The function accepts an integer array and performs an **IN-PLACE** sort using the MaxHeap functions above.
  - Precondition(s):
    - Size of int array array is greater than or equal to count
  - Postcondition(s):
    - Sorted: for each index $[0, \ldots, i, i+1, \ldots, n]$, array$[i] \leq$ array$[i+1]$

Late assignments will lose 25% per day late, with no assignment begin accepted after 4 days (100% reduction in points).

There are up to seven (7) points for correctness:

1. 1 POINT: Compilation without errors or warnings.
2. 1 POINT: Correctly implemented Add function for binary MaxHeap.
3. 1 POINT: Correctly implemented Remove function for binary MaxHeap.
4. 2 POINTS: Correctly implemented MaxHeapify function for binary MaxHeap.
5. 2 POINTS: Correctly implemented HeapSortAsc function for binary MaxHeap.

Additionally, there is a 25% penalty for compilation warnings and a 25% penalty for incorrect directory structure.

The last three (3) points are earned by correct styling. You should be using CppLint to determine correctness of style. There is **NO** reason to not earn these points.

One final note: You may not use the **algorithm** library. Notice in the provided test file I check for it.

The End.