

# Lab 04 - Heap & Priority Queue

## Instructions:

- A ‘priority queue’ is a data structure based on the principle that each element is associated with a priority, and the oldest highest-priority element is served first, regardless of its insertion order.

Priority queues have numerous real-world applications—such as task scheduling, event simulation, and pathfinding—and typically support four standard operations:

- `enqueue()` - adds item to the queue.
- `dequeue()` - removes the highest-priority (and oldest, if tied) item from the queue.
- `peek()` - returns, without removing, the highest-priority item in the queue.
- `empty()` - checks whether the queue is empty.

A priority queue can be efficiently implemented using a heap, which maintains a partial order to allow fast insertion and deletion. When implemented with a heap:

- `enqueue()` inserts the element at the end of the heap, then repeatedly swaps it with its parent until the heap property is restored (it either becomes the root or its parent has a higher or equal priority).
- `dequeue()` swaps the root with the last element, removes the last element, then heapifies the heap to restore order.
- `peek()` returns the root element of the heap.
- `empty()` checks whether the heap’s size is zero.
- The objective is to design the `enqueue()` algorithm for a priority queue where elements with lower key values have higher priority, effectively constructing the queue as a min-heap.
- Your source code must compile successfully and may only include the following libraries: ‘`iostream`’, ‘`iomanip`’, ‘`string`’, and ‘`Utils.h`’. No credit will be given if additional libraries are used. Documentation for the `Heap` and `Pair` classes is available in the lab directory.
- A cumulative task will not receive credit if the required previous tasks are not completed.
- Your submissions must be submitted to the GitHub repository in the Lab04 directory.
- Cheating of any kind is prohibited and will not be tolerated.
- **Violating or failing to follow any of the rules above will result in an automatic zero (0) for the lab.**

## Grading

Task	Maximum Points	Points Earned
1	1	
2	1	
3	1	
4	1	
5	1	
<b>Total</b>	<b>5</b>	

Note: solutions will be provided for tasks colored blue only.

## Task 1

- Create a text file named ‘`pseudocode.txt`’ that includes the pseudocode for the `Enqueue()` function for a priority queue described in the instructions above. The algorithm should take a heap and a pair as inputs.

## Task 2

- Create a text file named ‘`runtime.txt`’ that constructs a runtime and calculates the runtime function of your algorithm from Question 1.

## Task 3

- Create a header file named ‘`Algorithm.h`’ that includes ‘`Utils.h`’ and defines a C++ function of your pseudocode from Question 1. The inputs of the function must be a `Heap` object reference and a `Pair` object.

## Task 4

- Create a text file named ‘`simulate.txt`’ to simulate the execution of your algorithm from Question 1 for the following invocations starting with an empty heap:

```
Enqueue(A, (3, 'A')), Enqueue(A, (4, 'B')), Enqueue(A, (3, 'C')), Enqueue(A, (2, 'D')),  
Enqueue(A, (1, 'E')), Enqueue(A, (2, 'F')), Enqueue(A, (1, 'G'))
```

For each invocation, simulate the changes or focuses in the heap for each iteration of the loop and the outcome of the heap.

### Example:

For the invocations `Enqueue(A, (2, 'N'))`, `Enqueue(A, (1, 'E'))`, and `Enqueue(A, (1, 'W'))`, the simulations would be

- `Enqueue(A, (2, 'N'))`
  1. `[(2, 'N')]`
- `Enqueue(A, (1, 'E'))`
  1. `[(2, 'N'), (1, 'E')]`
  2. `[(1, 'E'), (2, 'N')]`
- `Enqueue(A, (1, 'W'))`
  1. `[(1, 'E'), (2, 'N'), (1, 'W')]`
  2. `[(1, 'E'), (1, 'W'), (2, 'N')]`

## Task 5

- Create a C++ file named ‘`test.cpp`’ that includes ‘`Algorithm.h`’ and verifies the outputs obtained in Question 4.

## Extra Credit

- Create the files
  1. ‘`pseudoExtra.txt`’ that provides the pseudocode of `Dequeue()` for the priority queue discussed in the instructions,

2. ‘timeExtra.txt’ that constructs a runtime table and calculates the runtime function of your pseudocode from ‘pseudoExtra.txt’,
3. ‘algorExtra.h’ that defines a C++ function of your pseudocode from ‘pseudoExtra.txt’. The function must use a *Heap* object reference as an input.

(1 point)