CMSC 123 LAB EXERCISE 7 — Total: 30 points

Priority Queues

In this exercise, you will be implementing the **Priority Queue** ADT using a **sorted** and **unsorted** list.

PriorityQueue:

- A data structure that stores **Entry** objects (key-value pairs), that follows the **Smallest Key-First Out** principle
  - For **Entry** objects, use the **getKey**( ) and **getValue**( ) methods
- The **key** is used as the **priority** level, and the entry with **smallest key** has the highest priority
- Priority Queue **methods**:
  - **size**( ) → returns the number of entries in the PQ
  - **isEmpty**( ) → returns true if PQ is empty (no elements)
  - **insert**(key, value) → add a new Entry; where to add Entry is implementation-dependent
  - **removeMin**( ) → remove Entry with smallest key, and return it
  - **min**( ) → return Entry with smallest key, without removing it

UnsortedPQ:

- Using only the three **properties** below, implement the PQ interface using an **unsorted DLL**:
  - **DLLNode<Entry> headGuard** → guard front of PQ; exists even if PQ is empty
  - **DLLNode<Entry> tailGuard** → guard back of PQ; exists even if PQ is empty
  - **int size** → keeps track of the PQ size (increase / decrease accordingly)
- Since the DLL is **unsorted**, you can choose where to insert your entries
- Implement the **findMin**( ) method → looks for the DLLNode<Entry> with minimum key
- Use the findMin( ) method in implementing removeMin( ) and min( )
- Use the ff. **DLLNode methods**: *getElement*, *setElement*, *getNext*, *setNext*, *getPrev*, *setPrev*

SortedPQ:

- Using only the three **properties** below, implement the PQ interface using a **sorted DLL**:
  - **DLLNode<Entry> headGuard** → guard front of PQ; exists even if PQ is empty
  - **DLLNode<Entry> tailGuard** → guard back of PQ; exists even if PQ is empty
  - **int size** → keeps track of the PQ size (increase / decrease accordingly)
- Since the DLL is **sorted,** you have to find the **correct position** to **insert** nodes into to maintain the sortedness of the DLL
- The advantage for a sorted list is that it will be easy to implement min( ) and removeMin( )
- Use the ff. **DLLNode methods**: *getElement*, *setElement*, *getNext*, *setNext*, *getPrev*, *setPrev*

Checker:

- Test your UnsortedPQ and SortedPQ implementations using the checker **Exercise7.java**
- The checker tests your code on various scenarios and method calls to check if your implementation is working correctly
- *Usage*: java Exercise7 unsorted or java Exercise7 sorted

Scoring:

- **(15 pts) UnsortedPQ**
  - **3 pts** - insert( )
  - **5 pts** - findMin( )
  - **5 pts** - removeMin( )
  - **2 pts** - min( )
- **(10 pts) SortedPQ**
  - **6 pts** - insert( )
  - **4 pts** - removeMin( ) / min( )
- **(5 pts) Checker**