CMSC 123 LAB EXERCISE 8 **Total: 40 points**
Maps and Dictionaries *Individual / By Pair*

In this exercise, you will be implementing the **Map** ADT using an **unsorted doubly-linked list,** and the **Dictionary** ADT using a **sorted array**. You can work on this exercise individually or as a pair.

Map

- A data structure that stores **Entry** objects (key-value pairs), where the **keys** are **unique**
  - Use **getKey**( ) → *String* and **getValue**( ) → *Object* methods for **Entry** objects
- Map **methods:**
  - int **size**( ) → returns the number of entries in the map
  - boolean **isEmpty**( ) → returns true if the map is empty (no entries)
  - Object **get**(key) → returns the associated value if key is found, otherwise returns null
  - Object **put**(key, value) → if key doesn't exist in the map, adds new entry and returns null, otherwise, it overwrites the existing entry, and returns the old value
  - Object **remove**(key) → if key exists, remove the associated entry, and return the value, otherwise, return null
  - Iterator<String> **keys**( ) → an iterator that allows you to iterate through the map's keys
  - Iterator<Object> **values**( ) → an iterator that allows you to iterate through the map's values
  - Iterator<Entry> **entries**( ) → an iterator that allows you to iterate through the map's entries

UnsortedDLLMap

- Using only the three **properties** below, implement the **map** interface using an **unsorted DLL**:
  - **DLLNode<Entry> headGuard** → guard front of map; exists even if map is empty
  - **DLLNode<Entry> tailGuard** → guard back of map; exists even if map is empty
  - **int size** → keeps track of the map's size (increase / decrease accordingly)
- Since the DLL is **unsorted**, you can choose where to insert your entries
- Implement the **findNode**( ) method, using **linear search** → look for a given key by checking each Entry in the map **sequentially,** and return the associated **DLLNode**
- Use the findNode( ) method in implementing get( ), put( ), and remove( )
- Use the ff. **DLLNode methods**: *getElement*, *setElement*, *getNext*, *setNext*, *getPrev*, *setPrev*
- The codes for the **Iterator** methods: keys( ), values( ), entries( ) should be **similar** → they only differ in what they return (String key / Object value / Entry entry).

Dictionary

- A data structure that stores **Entry** objects (key-value pairs), where the **keys** can be **repeated**
  - Use **getKey**( ) → *String* and **getValue**( ) → *Object* methods for **Entry** objects
- Dictionary **methods:**
  - int **size**( ) → returns the number of entries in the dictionary
  - boolean **isEmpty**( ) → returns true if the dictionary is empty (no entries)
  - Entry **get**(key) → if key exists, return *any* Entry with the given key, otherwise return null
  - Entry **put**(key, value) → inserts new Entry and returns it; don't need to worry about overwriting existing key, as duplicate keys are allowed
  - void **remove**(entry) → removes the given Entry if it is found
  - Iterator<Entry> **entries**( ) → an iterator that allows you to iterate through the dictionary's entries
  - Iterator<Entry> **getAll**(key) → an iterator that allows you to iterate through all the entries that match the given key

SortedArrayDictionary

- Using only the two **properties** below, implement the **dictionary** interface using a **sorted array:**
  - **Entry[ ] array** → array that will hold Entry objects; keep array sorted at all times
  - **int size** → keeps track of the dictionary's size (increase / decrease accordingly)
- Since the array is **sorted**, you have to find the **correct index** to **insert** an Entry into to maintain the sortedness of the array
- Implement **findIndex**( ), using **binary search** → look for a given key using a **divide-and-conquer** approach, and return its index; if key is not found, and **includeInsert** flag = true, return the correct index to insert the entry into, to keep the array sorted
- Use the findIndex( ) method in implementing get( ), put( ), and remove( )
- In the **put**( ) method, move some entries **forward**, if necessary
- In the **remove**( ) method, move some entries **backward**, if necessary
- The codes for the **Iterator** methods: entries( ) and getAll( ) should be **similar** → they only differ in the values for startIndex and limit.

Checker:

- Test your UnsortedDLLMap and SortedArrayDictionary implementations using the checkers, **Exercise8M.java** (Map) and **Exercise8D.java** (Dictionary)
- The checker tests your code on various scenarios and method calls to check if your implementation is working correctly

Scoring:

- **(18 pts) UnsortedDLLMap**
  - **4 pts** - findNode( )
  - **2 pts** - get( )
  - **4 pts** - put( )
  - **3 pts** - remove( )
  - **5 pts -** keys( ) / values( ) / entries( )
- **(20 pts) SortedArrayDictionary**
  - **6 pts** - findIndex( )
  - **2 pts** - get( )
  - **4 pts** - put( )
  - **3 pts** - remove( )
  - **5 pts -** entries( ) / getAll( )
- **(2 pts) Checker**