Lab Assignment #2 [Weight: ~5% of the Course Grade]

Topic: Linked List Data Structure

- **For this assignment, you must work in pairs or — exceptionally — in teams of three.**
- **Include the name and ID for each group member in your files.**
- **For C/C++ implementation, you must separate class header and class implementation. You may not use other languages for this assignment.**
- **Please submit your completed assignment before the dropbox closes on LEARN.**

Drone Fleet Management as an ADT

Our company is in a possession of a small fleet of drones that we would like to make available to customers. We are tasked with programming a data structure for storage of drone records.

Submit three files: **lab2_drones_manager.cpp, lab2_drones_manager.hpp,** and **lab2_drones_manager_test.hpp**. Do not modify the signatures for any of the functions listed below.

Step0. [see lab2_drones_manager.hpp]

We are providing you with the implementation of the private **DroneRecord** class. Each drone is to be recorded as an instance of this class. Each record has a unique drone ID, which is stored as an unsigned integer. In addition, each **DroneRecord** also stores drone type, manufacturer, description, range, battery type, and year bought; range and year bought are to be stored as unsigned integers while others are to be stored as strings.

We are also providing you with the declaration for the **DronesManager** class. This class will be used as a container to handle **DroneRecord** objects, and will be based on a doubly linked list implementation of List ADT. **DronesManager** includes relevant member attributes, and corresponding accessor and mutator functions.

Step1. [see lab2_drones_manager.hpp]

Implement all of the methods for the class **DronesManager** that are listed below.

class DronesManager {…} [header already defined in lab2_drones_manager.hpp; provide your definitions in lab2_drones_manager.cpp]

```
// EXPLICIT CONSTRUCTOR AND DESTRUCTOR
// PURPOSE: Creates a new empty DronesManager
DronesManager();

// PURPOSE: Destroys this instance and frees up all dynamically allocated memory
~DronesManager();
```

```cpp
    // PURPOSE: Comparison operator to compare two DroneRecord instances
    friend bool operator==(const DronesManager::DroneRecord& lhs,
                           const  DronesManager::DroneRecord& rhs);

    // PURPOSE: Setup DronesManagerTest as friend so tests have access to private variables
    friend class DronesManagerTest;
```

// ACCESSORS
```cpp
    // PURPOSE: Returns the number of elements in the list
    unsigned int get_size() const;

    // PURPOSE: Checks if the list is empty; returns true if the list is empty, false otherwise
    bool empty() const;

    // PURPOSE: Returns the value at the given index in the list
    // if index is invalid, returns last element; if the list is empty, returns DroneRecord(0)
    DroneRecord select(unsigned int index) const;

    // PURPOSE: Searches for the given value, and returns the index of this value if found
    // if not found, returns the size of the list; if the list is empty, returns 0
    unsigned int search(DroneRecord val) const;

    // PURPOSE: Prints all the elements in the list to the console
    void print() const;
```

// MUTATORS
```cpp
    // PURPOSE: Inserts a value into the list at a given index; the list is not sorted
    // if the index is invalid, insertion is rejected
    bool insert(DroneRecord val, unsigned int index);

    // PURPOSE: Inserts a value at the beginning of the list; the list is not sorted
    bool insert_front(DroneRecord val);

    // PURPOSE: Inserts a value at the end of the list; the list is not sorted
    bool insert_back(DroneRecord val);

    // PURPOSE: Deletes a value from the list at the given index
    bool remove(unsigned int index);

    // PURPOSE: Deletes a value from the beginning of the list
    bool remove_front();

    // PURPOSE: Deletes a value at the end of the list
    bool remove_back();

    // PURPOSE: Replaces value at the given index with the given value; the list is not sorted
    // if the index is invalid, replacement is rejected
    bool replace(unsigned int index, DroneRecord val);

    // PURPOSE: Reverses the linked list
    bool reverse_list();
```

Step2. [see lab2_drones_manager_test.hpp]

Implement all of the test case methods in the **DronesManagerTest** class that are listed below. If the code is already provided, implement a different version of the test case on your own. The test cases will be graded.

class DronesManagerTest {…}
[declared and implemented in lab2_drones_manager_test.hpp;
test runner in lab2_main.cpp]

```
// TEST CASES
// PURPOSE: New empty list is valid
bool test1();

// PURPOSE: insert_front() and insert_back() on zero-element list
bool test2();

// PURPOSE: select() and search() work properly
bool test3() ;

// PURPOSE: remove_front() and remove_back() on one-element list
bool test4();

// PURPOSE: replace() and reverse_list() work properly
bool test5();

// PURPOSE: insert_front() keeps moving elements forward
bool test6();

// PURPOSE: inserting at different positions in the list
bool test7();

// PURPOSE: try to remove too many elements, then add a few elements
bool test8();

// PURPOSE: lots of inserts and deletes, some of them invalid
bool test9();

// PURPOSE: lots of inserts, reverse the list, and then lots of removes until empty
bool test10();
```