Assignment 1, Part 3 of 3
Doubly Linked List Implementation

Instructor: Homeyra Pourmohammadali
MTE140 - Data Structures and Algorithms
Winter 2021 (Online)
University of Waterloo

Due: 11:00 PM, Friday,Feb 12, 2021

Purpose of this assignment

In this assignment, you will practice your knowledge about **linked list** by implementing a data type called doubly linked list. The header file `doubly-linked-list.h`, which is explained below, provides the structure of the `DoublyLinkedList` class with declarations of member functions. Do not modify the signatures of any of these functions. You need to implement all of the public member functions listed in `doubly-linked-list.cpp`. The memory used by this list will change dynamically as nodes are inserted and removed from the list.

Instruction

Sign in to Gitlab to verify that you have a project set up for your Assignment 1 (a1) at https://git.uwaterloo.ca/mte140/mte140-1211-a1-WATIAM_ID with the following files.

```
YOUR-WATIAM-ID
├── CMakeLists.txt
├── README.md
├── user.yml
├── doubly-linked-list.cpp
├── doubly-linked-list.h
├── sequential-list.cpp
├── sequential-list.h
└── test.cpp
```

For this part of assignment, you only need to modify `doubly-linked-list.cpp`. And you should have updated `user.yml` in the previous part. You can design your onw test case and code in `test.cpp`. It is optional and we will not grade this file.

You can use the same procedures in Activity 0 to pull, edit, build, commit, and push your repo.

Description

The details of the header file `doubly-linked-list.h` are as follows:
`DataType` defines the kind of data that the list will contain. Being public, it can be accessed directly as `DoublyLinkedList::DataType`.

Member variables:
`Node`: This is a structure declaration. Node has been declared inside the class as private, to make sure users do not access nodes directly. Node contains the following member variables:

- `value`: Value of the node.
- `ext`: Address of the next element in the list.
- `rev`: Address of the previous element in the list.

`head_`: Pointer to the first node of the list.
`tail_`: Pointer to the last node of the list.
`size_`: Number of elements (nodes) in the list.

Constructor and Destructor:
`DoublyLinkedList()`: Initializes an empty doubly linked list.
`DoublyLinkedList()`: Destructor, which frees all dynamically allocated memory, if any.
`DoublyLinkedList(const DoublyLinkedList& rhs)`: Copy constructor, which copies one list `rhs` to another list. For this assignment, you don't have to implement it. Just leave it blank in private section.
`DoublyLinkedList& operator=(const DoublyLinkedList& rhs)`: Assignment operator for the operations of the form list1 = list2. For this assignment, you don't have to implement it. Just leave it blank in private section.

Constant member functions:
Constant member functions are those function which do not modify class member variables.
`unsigned int size() const`: Returns the number of elements `nodes` in the list.
`unsigned int capacity() const`: Returns the maximum number of elements the list can hold.
`bool empty() const`: Returns true if the list is empty, false otherwise.
`bool full() const`: Returns true if the list is at capacity, false otherwise.
`DataType select(unsigned int index) const`: Returns the value at the given index in the list. If index is invalid, return the value of the last element.
`unsigned int search(DataType val) const`: Searches for the given value, following the order from head to tail of the list, and returns the index of this value if found (for the first time the value is found). Returns the size of the list if no such value can be found in the list.
`void print() const`: Prints all elements in the list to the standard output.
`Node* getNode(unsigned int index) const`: Returns a pointer to the node at index.

Non-constant member functions:
These functions can potentially modify member variable of the class.
`bool insert(DataType val, unsigned int index)`: Inserts a value into the list at a given index. Returns true if successful and false otherwise.

`bool insert_front(DataType val)`: Inserts a value at the beginning of the list. Returns true if successful and false otherwise.
`bool insert_back(DataType val)`: Inserts a value at the end of the list. Returns true if successful and false otherwise.
`bool remove(unsigned int index)`: Deletes the value from the list at the given index.
`bool remove_front()`: Deletes the value from the beginning of the list. Returns true if successful and false otherwise.
`bool remove_back()`: Deletes the value at the end of the list. Returns true if successful and false otherwise.
`bool replace(unsigned int index, DataType val)`: Replaces the value at the given index with the given value.

Note: All indexes must start from 0.

Marking

We will try different inputs and check your output. We will only test your program with syntactically and semantically correct inputs.
Part 3 counts 50% of Assignment 1, which is 50 points in total.

Your program runs and does not crash during the test: + 20
Passes Test Cases: + 3 each, in total of 30