CSE250 Project #1

Sheng Liu

June 2021

1 Introduction

This project ask you to implement **doubly-linked list**, *i.e.*, a type of linked data structure that consists of a set of sequentially linked records called nodes. Each node contains three fields: two link fields (pointers to the previous and to the next node in the sequence of nodes) and one data field. The beginning and ending nodes' previous and next links, respectively, point to some kind of terminator, *i.e.*, `nullptr` (in our case).

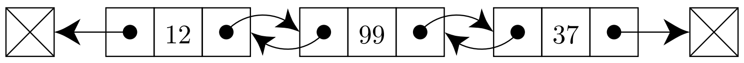


Figure 1: An illustration of doubly-linked-list.

2 Details

A header file named "List.h" and a cpp file named "List.cpp" are provided (can be downloaded from Piazza). The header file contains declarations for member functions and member variables of two classes: `Node` and `DoublyLinkedList`. We have implemented the `Node` class for you in the cpp file, which means that you only need to implement the `DoublyLinkedList` class. To make this project a bit easier, we assume that data records stored in the nodes of doubly-linked lists are of type `int`.

For the `DoublyLinkedList` class. you are asked to implement its five member functions and one constructors.

- `DoublyLinkedList()`: a constructor that takes no argument. We have implemented it for you.
- `DoublyLinkedList(int arr[], int size)`: a constructor that takes as input an array of integers and the size of the array. The size of the array may be 0, *i.e.*, the array might be an empty one. For example, the list shown in Fig. 1 can be constructed via:

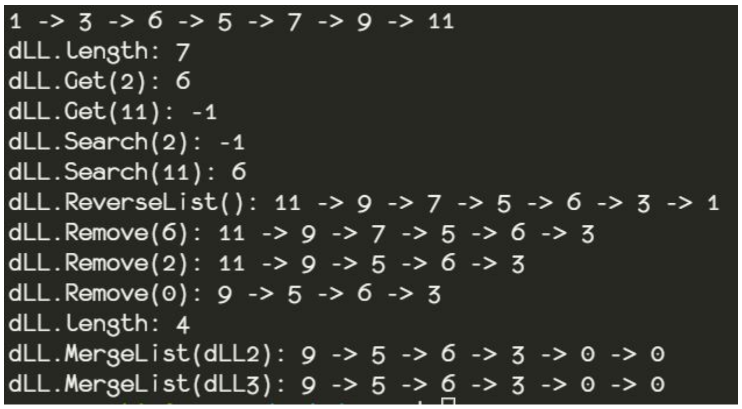
  `int arr[] = {12, 99, 37};`

```
DoublyLinkedList list = DoublyLinkedList(arr, 3);
```

- `Insert(int idx, int v)`: inserts an integer `v` at index `idx`. For example, if we call `list.Insert(3, 28)`, 28 should be inserted to the after 37, *i.e.*, the last element in the list. If we call `list.Insert(0, 28)`, 28 should be inserted to the before 12, *i.e.*, the first element in the list. Do **not** do anything, if `idx` is not valid, *i.e.*, if `idx` < 0 or `idx` $> n$ (n is the number of nodes in the list).
- `Remove(int idx)`: removes the $i-$th node in the list, $i =$`idx`$+1$. For example, if we call `list.Remove(2)`, 37 should be removed. If we call `list.Remove(0)`, 12 should be removed. Do **not** do anything, if `idx` is not valid, *i.e.*, if `idx` < 0 or `idx` $> n - 1$.
- `Get(int idx)`: returns the **value** stored in the $i-$th node in the list, $i =$`idx`$+1$. For example, `list.Get(1)` returns 99. It returns -1 if `idx` is not valid, *i.e.*, if `idx` < 0 or `idx` $> n - 1$.
- `Search(int v)`: searches for a given integer `v` in the list and returns the index of the node that stores `v`. For example, `list.Search(99)` returns 1. If `v` is not stored in any node of the list, it returns -1. If multiple nodes stores `v`, it returns the smallest index.
- `ReverseList()`: reverses the list. For example, if we call `list.ReverseList()`, the list becomes 37 (1st record), 99 (2nd record), 12 (3rd record). Note: remember to properly modify the two pointers `head` and `tail`.
- `MergeList(DoublyLinkedList *list2)`: merges `list2` into the list. For example, suppose `list2` stores 12, 99. After calling `list.MergeList(list2)`, `list` stores five records 12, 37, 99, 12, 99 (listed in order, from `head` to `tail`). `list2` may be an empty list.
- `PrintList()`: prints the values stored in all the nodes of the list in order (from `head` to `tail`). Two values should be separated by a space.
- `length`: stores the length of the list, *i.e.*, number of nodes in the list.
- `head, tail`: pointers to the first and the last nodes of the list. They should be set to `nullptr` if the list is empty.

3 Submission

As mentioned earlier, you **only** need to submit the cpp file named "List.cpp" to AutoLab. Do **NOT** modify the code we provided in the cpp file or using any additional function defined in C++ Standard Library. As your projects will be graded automatically, modifying the provided code may cause our grading script to fail. If this happens, the grade will be **0**. Grades for the projects that use additional std functions will also be **0**. Our grading script will call the functions



```
1 -> 3 -> 6 -> 5 -> 7 -> 9 -> 11
dLL.Length: 7
dLL.Get(2): 6
dLL.Get(11): -1
dLL.Search(2): -1
dLL.Search(11): 6
dLL.ReverseList(): 11 -> 9 -> 7 -> 5 -> 6 -> 3 -> 1
dLL.Remove(6): 11 -> 9 -> 7 -> 5 -> 6 -> 3
dLL.Remove(2): 11 -> 9 -> 5 -> 6 -> 3
dLL.Remove(0): 9 -> 5 -> 6 -> 3
dLL.Length: 4
dLL.MergeList(dLL2): 9 -> 5 -> 6 -> 3 -> 0 -> 0
dLL.MergeList(dLL3): 9 -> 5 -> 6 -> 3 -> 0 -> 0
```

you implement in the cpp file and check whether the functions do what they are asked to.

4 An Example

To help you understand how our grading script call the functions you implement in "List.cpp". We provide an **example**, named "main.cpp" (can be downloaded from Piazza). The desired output for the provided is example is shown below. Note that the format of the printing function is a bit different. You should follow the format specified in this file.