Middle East Technical University | Department of Computer Engineering

CENG 213

Data Structures

Fall 2019-2020

Programming Assignment 1

Due date: 25.10.2019 23:55

1 Objectives

In this assignment you are expected to implement a doubly linked list data structure with dummy header and tail nodes. In this data structure each node will have pointers to both the previous and the next node. The head pointer in the data structure will point to a node that have no data in it and point to the first informative node as its next. The tail pointer points to the last node (dummy, has no data) which points to the last informative node as its previous. The details of the structure is explained further in the following sections. You will use this specialized linked list structure to represent a simple web browser history system and implement some functions on it.

2 Linked List Implementation (70 pts)

The linked list data structure used in this assignment is implemented as the class template LinkedList with the template argument T, which is used as the type of the data stored in the nodes. The node of the linked list is implemented as the class template Node with the template argument T, which is the type of the data stored in nodes. Node class is the basic building block of the LinkedList class. LinkedList class has two Node pointers (head and tail) in its private data field, which point to the first and the last nodes of the linked list. However, please note that, **head and tail nodes are dummy nodes**, which means they hold no informative data in them but they point to the actual first and last nodes respectively.

The LinkedList class has its definition and implementation in LinkedList.hpp file and the Node class has its in Node.hpp file.

2.1 Node

Node class represents nodes that constitute linked lists. A Node keeps two pointers (namely prev and next) to its previous and next nodes in the list, and the data variable of type T (namely element) to hold the data. The class has three constructors (including a copy constructor), and the overloaded output operator. They are already implemented for you. You should not change anything in file Node.hpp.

2.2 LinkedList

LinkedList class implements a doubly linked list data structure with the dummy head and tail pointers. Previously, data members of LinkedList class have been briefly described. Their use will be elaborated in the context of utility functions discussed in the following subsections. You must provide implementations for the following public interface methods that have been declared under indicated portions of LinkedList.hpp file.

2.2.1 LinkedList()

This is the default constructor. You should make necessary initializations in this function.

2.2.2 LinkedList(const LinkedList &obj);

This is the copy constructor. You should make necessary initializations, create new nodes by copying the nodes in given obj and insert new nodes into the linked list.

2.2.3 ∼ LinkedList();

This is the destructor. You should deallocate all the memory that were allocated for the linked list.

2.2.4 Node<T> *getHead() const;

This function should return the head pointer of the linked list.

2.2.5 Node<T> *getTail() const;

This function should return the tail pointer of the linked list.

2.2.6 Node<T> *getFirstNode() const;

This function should return pointer to the first data node. If the list is empty, this function should return the tail node.

2.2.7 int getNumberOfNodes();

This function should return an integer that is the number of nodes in the linked list. Do not count the head and tail nodes.

2.2.8 bool isEmpty();

This function should return true if the linked list is empty (i.e. there exists no nodes in the linked list other than head and tail). If it is not empty, it should return false.

2.2.9 void insertAtTheFront(const T &data);

You should create a new node with given data and insert it at the front of the linked list after the dummy header node.

2.2.10 void insertAtTheEnd(const T &data);

You should create a new node with the given data and insert it at the end of the linked list as the last node before the tail node. Do not forget to make necessary pointer modifications.

2.2.11 void insertAfterGivenNode(const T &data, Node<T> *prev);

You should create a new node with given the data T and insert it after the node pointed by the *prev*.

2.2.12 void removeNode(Node<T> *node);

You should remove the given node node from the linked list. If the given node node is not in the linked list, do nothing.

2.2.13 void removeAllNodes();

You should remove all the nodes in the linked list, except for the dummy head and tail nodes.

2.2.14 Node<T> *findNode(const T &data);

You should search for the first occurance of a node in the linked list with the data same as the given data and return a pointer to that node. You can use the operator== to compare two T objects. If there exists no such node in the linked list, you should return NULL.

2.2.15 void printAllNodes();

All nodes in the linked list are printed from head to tail direction. Each node is printed on a single line.

2.2.16 void printReversed();

All nodes in the linked list are printed from tail to head direction using prev pointers. Each node is printed on a single line.

2.2.17 LinkedList &operator=(const LinkedList &rhs);

This is the overloaded assignment operator. You should remove all nodes in the linked list and then create new nodes by copying the nodes in given the rhs and insert new nodes into the linked list.

3 Web History Implementation (30 pts)

As mentioned before, you will use the linked list data structure to simulate a basic web browser history functionality. For this assignment, we will have some assumptions for the web history. First of all, each history element will be represented by two fields, namely, the url of the page and the last visited time as the timestamp. The next assumptions are that, there will be no repeating urls in the history and timestamp will always be positive. As the final assumption, you will never be asked to update the last visited time of an url (you will not visit the same page more than once).

The web history in this assignment is implemented as the class WebHistory. WebHistory class has a LinkedList object (namely history) with the type Tab in its private data field. Tab class represents the visited and closed tabs/pages via the browser. It is the type of the data stored in nodes of the linked list.

The WebHistory class has its definition in WebHistory.hpp file and its implementation in WebHistory.cpp file. The Tab class has its definition in Tab.hpp file and its implementation in Tab.cpp file.

3.1 Tab

Tab class represents a history item in the browser history list. A Tab keeps the **url** string and **timestamp** variable of type int to hold the data related with the tab. The class has three constructors (including a copy constructor), getter and setter functions, the overloaded output operator, and the overloaded equality/inequality check operators. They are already implemented for you. You should not change anything in files Tab.hpp and Tab.cpp.

3.2 WebHistory

In WebHistory class, there is a single member variable named as **history**, which is a LinkedList of Tab nodes. Information of all tabs will be stored in this linked list in decreasing order of timestamps. All member functions should utilize this linked list to operate as described in the following subsections. Default constructor and the constructor that takes tab information as a string and populates the tabs to the linked list have already been implemented for you. Do not change those implementations. In webHistory.cpp file, you need to provide implementations for the following functions declared under webHistory.hpp header to complete the assignment. You should not change anything in file webHistory.hpp.

3.2.1 WebHistory();

This is the default constructor. It is already implemented.

3.2.2 WebHistory(std::string history);

This is the constructor that takes a series of url and timestamps as a string and populates the "tabs" into the linked list. The string is in the form of "url-1 timestamp-1 | url-2 timestamp-2 | url-3 timestamp3 | ... | url-n timestamp-n". It is already implemented.

3.2.3 WebHistory(std::string url, int timestamp);

This is the constructor that takes a single url and a timestamp (history instance) and creates a history linked list with a single tab.

3.2.4 void printHistory();

This function prints the history in a specific format. It is already implemented. You can track the format of its output from the code.

3.2.5 void insertInOrder(Node<Tab> *newPage);

This function inserts the given tab (linked list node) to the history so that the history list should stay in decreasing order of timestamp values. The constructor also uses this function which means that you can assume that the history list is always sorted with respect to timestamps. You should find the correct place for the new page and make necessary pointer adjustments.

3.2.6 void goToPage(std::string url, int timestamp);

This function does the same thing with *insertInOrder* function. However, this time it takes *url* and *timestamp* as parameters. Therefore, you should create a new tab with the given parameters and then take necessary actions as in the *insertInOrder*.

3.2.7 void clearHistory();

This function removes all tabs from the history. In other words, only the head and tail nodes will remain in the linked list after this function terminates.

3.2.8 void clearHistory(int timestamp);

This function finds the first tab with a timestamp less than or equal to the given timestamp and removes all the tabs comes after it from the history. For example, if the given timestamp is 4 and the history is "head-6-5-3-2-1-tail", the resulting history should be "head-6-5-tail".).

3.2.9 WebHistory operator+(const WebHistory &rhs) const;

This is the overloaded + operator. You should create a new WebHistory object with the tabs of the first WebHistory and merge it with nodes of second WebHistory (rhs). The resulting history should again be sorted in regard to timestamp values.

3.3 WebHistory - Bonus Parts

3.3.1 int timesVisited(std::string pageName);

This function should return how many times the given page is visited. Please note that, the urls in the tabs are full links (i.e. `https://www.youtube.com/watch?v=njTh_OwMljA`) whereas the *pageName* is just the web page's name (youtube in this case). Necessary helper function is given in the Tab class.

3.3.2 std::string mostVisited();

This function returns the name of the most visited page's name. As in the previous function, we are not dealing with the full page links. Instead, you should strip the page name using the given helper function in Tab class and count the occurrences with stripped names. In other words, you should count all youtube links in the history even if their full links are different.

4 Driver Programs

To enable you to test your LinkedList and WebHistory implementations, two driver programs, main_linkedlist.cpp and main_browser.cpp are provided. Their expected outputs are also provided in output linkedlist.txt and output browser.txt files, respectively.

5 Regulations

1. **Programming Language:** You will use C++.
2. External libraries other than those already included are not allowed.

6 Submission

- Submission will be done via CengClass (cengclass.ceng.metu.edu.tr).
- Do not write a *main* function in any of your source files.
- A test environment will be ready in CengClass.

- You can submit your source files to CengClass and test your work with a subset of evaluation inputs and outputs.
- Additional test cases will be used for evaluation of your final grade. So, your actual grades may be difierent than the ones you get in CengClass.
- Only the last submission before the deadline will be graded.