CENG 707 Ceng213 - Data Structures
Programming Assignment 1 : Matrices via Linked Lists

Fall 2021 Spring 2020

1 Objectives

In this first programming assignment, you are first expected to implement a *doubly linked list* data structure, in which each node will contain the data and two pointers to the previous and the next nodes. The data structure will include a head and a tail pointer that points to the first and the last nodes in the list. The details of the structure are explained further in the following sections. Then, you will use this specialized linked list structure to represent *matrices* and implement some functions on them.

Keywords: *Data Structures, Linked List, Matrices*

2 Linked List Implementation (60 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 (in its private data field) two `Node` pointers (namely `head` and `tail`) which point to the first and the last nodes of the linked list, and an integer (namely `size`) which keeps the number of nodes in the list.

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 `data`) 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 `head` and the `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 you were allocated before.

2.2.4 `int getSize() const;`

This function should return an integer that is the number of nodes in the linked list.

2.2.5 `bool isEmpty() const;`

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

2.2.6 `void insertAtTheFront(const T &data);`

You should create a new node with given `data` and insert it at the front of the linked list as the first node. Don't forget to make necessary pointer, head-tail, and size modifications.

2.2.7 `void insertAtTheEnd(const T &data);`

You should create a new node with given `data` and insert it at the end of the linked list as the last node. Don't forget to make necessary pointer, head-tail, and size modifications.

2.2.8 `void insertBeforeNode(const T &data, Node<T> *node);`

You should create a new node with given `data` and insert it before the given node `node` as its previous node. Don't forget to make necessary pointer, head-tail, and size modifications. If the given node `node` is not in the linked list, do nothing.

2.2.9 `void deleteNode(Node<T> *node);`

You should delete the given node `node` from the linked list. Don't forget to make necessary pointer, head-tail, and size modifications. If the given node `node` is not in the linked list, do nothing.

2.2.10 `void deleteAllNodes();`

You should delete all nodes in the linked list.

2.2.11 `Node<T> *getFirstNode() const;`

This function should return a pointer to the first node in the linked list. If the linked list is empty, it should return `NULL`.

2.2.12 `Node<T> *getLastNode() const;`

This function should return a pointer to the last node in the linked list. If the linked list is empty, it should return `NULL`.

2.2.13 `Node<T> *getNodeWithData(const T &data) const;`

You should search for the node in the linked list with the data same with 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`.

2.2.14 `void traverseAllNodes(Node<T> *node);`

Output of this function and the already implemented `traverseAllNodes()` function should have the same format. The only difference between these two functions is that while the traversal begins from the first node (`head`) in the already implemented function, it starts from the given node `node` and traverses all nodes circularly. You will not be asked to begin from a node that is not in the linked list.

2.2.15 `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 `rhs` and insert new nodes into the linked list.

3 Matrix Implementation (40 pts)

The matrices in this assignment is implemented as the class `Matrix`. `Matrix` class has (in its private data field) a `LinkedList` object (namely `matrix`) with the type `int`, and two integers (namely `numberOfRows` and `numberOfColumns`) which keeps the number of rows and columns in the matrix, respectively. `int` is the type of the data stored in nodes of the linked list.

The `Matrix` class has its definition in *Matrix.hpp* file and its implementation in *Matrix.cpp* file.

3.1 `Matrix`

In `Matrix` class, all member functions should utilize `matrix`, `numberOfRows`, and `numberOfColumns` member variables to operate as described in the following subsections. Default constructor and the constructor that takes elements of a matrix as a string and populates the elements to the linked list have already been implemented for you. Don't change those implementation. In *Matrix.cpp* file, you need to provide implementations for following functions declared under *Matrix.hpp* header to complete the assignment. You should not change anything in file *Matrix.hpp.*

3.1.1 `int getElementByIndex(int rowIndex, int columnIndex) const;`

This function should return the element of the matrix at given 0-based indexes (`rowIndex` and `columnIndex`). You may assume that the given indexes are valid indexes.

3.1.2 `void setElementByIndex(int rowIndex, int columnIndex, int element);`

This function should set the element of the matrix at given 0-based indexes (`rowIndex` and `columnIndex`) to the given integer value `element`.

3.1.3 `Matrix operator+(const Matrix &rhs) const;`

This is the overloaded `operator+` to implement matrix addition. You should create a new `Matrix` object by element-wise addition of the elements of the first matrix and the matrix `rhs`. You may assume that the dimensions of the both matrices are same.

3.1.4 `Matrix operator*(const int &rhs) const;`

This is the overloaded `operator*` to implement multiplication of matrix by integer scalar. You should create a new `Matrix` object by element-wise multiplication of the elements of the first matrix by the integer scalar `rhs`.

3.1.5 `Matrix kroneckerProduct(const Matrix &rhs) const;`

This is the function to implement Kronecker production. You should create a new `Matrix` object which is the Kronecker product of the first matrix and the matrix `rhs`. Details of Kronecker product (including a sample step by step application) can be found at the Wikipedia page "Kronecker product".

4 Driver Programs

To enable you to test your `LinkedList` and `Matrix` implementations, two driver programs, *main_linkedlist.cpp* and *main_matrix.cpp* are provided. Their expected outputs are also provided in *output_linkedlist.txt* and *output_matrix.txt* files, respectively.

5 Regulations

1. **Programming Language:** You will use C++.
2. **S**tandard Template Library is **not** allowed.
3. **E**xternal libraries other than those already included are **not** allowed.
4. **T**hose who do the operations (insert, delete, get, traverse) without utilizing the linked list will receive **0 grade**.
5. **T**hose who modify already implemented functions and those who insert other data variables or public functions and those who change the prototype of given functions will receive **0 grade**.

6. **T**hose who use STL vector or compile-time arrays or variable-size arrays (not existing in ANSI C++) will receive **0 grade**. Options used for g++ are "-ansi -Wall -pedantic-errors -O0". They are already included in the provided Makefile.

7. **Y**ou can add private member functions whenever it is explicitly allowed.

8. **Late Submission:** Each assignment will have a fixed duration of 10 days and every student has a total of 5 days for late submissions during which no points will be deducted. Your assignment will not be accepted if you used up all of the 5 late days.

9. **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations. Remember that students of this course are bounded to code of honor and its violation is subject to severe punishment.

10. **Newsgroup:** You must follow the Forum (`odtuclass.metu.edu.tr`) for discussions and possible updates on a daily basis.

6 Submission

- Submission will be done via CengClass (`cengclass.ceng.metu.edu.tr`). email to the instructor.
- Don't 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 different than the ones you get in CengClass.
  - Only the last submission before the deadline will be graded.