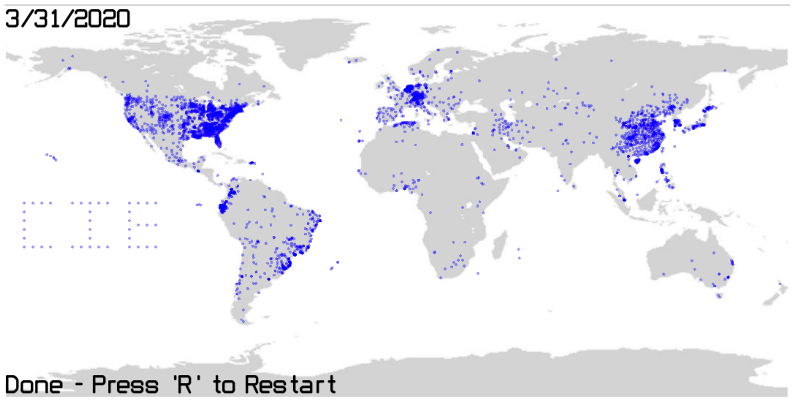
ENGR:2730 Computers in Engineering
Christensen, Spring 2020, Homework #7

Deadline: Friday, May 8, 2020 by 11:59pm
Early turn in bonus deadline: Wednesday, May 6, 2020 by 11:59pm

In this homework, you will implement a date-sorted, singly linked list class to keep track of COVID-19 confirmed cases. Your list will store the latitude, longitude and date of each confirmed case. You will write two new classes, **Node** and **MyList**. These classes must be implemented in their own files, i.e., Node.cpp, Node.h, MyList.cpp and MyList.h. We have provided you with a program that will use your Node and MyList classes to plot the daily confirmed COVID-19 cases across the world.



Getting Started:

1. Copy the "hw7" folder in the "Public/Homework/" directory and paste it into your own homework folder.
2. This program is initially set up with the graphics disabled. You must fully implement your Node and MyList classes and test them before enabling the graphics. **When you are finished with your homework**, you can change the unitTesting variable from true to false to enable the graphics and see you program in action.

Specifications:

1. **[25pts]** Implement and document a **Node** class.
   a. Your Node class must have exactly six private member variables named **float m_latitude**, **float m_longitude**, **int m_day**, **int m_month**, **int m_year and Node *m_nextPtr**. These variables must be defined in the exact order given.
   b. Your Node class must have an explicit constructor that takes six parameters that are listed in the exact order as part a. This constructor must initialize all six private member variables. The constructor must provide default values for each member variable.
   c. Implement getters and setters for all private member variables. The class prototypes for these methods must be named exactly as follows.

```
float getLatitude() const;
void setLatitude(float latitude);
float getLongitude() const;
void setLongitude(float longitude);
int getDay() const;
void setDay(int day);
int getMonth() const;
void setMonth(int month);
int getYear() const;
void setYear(int year);
Node *getNextPtr() const;
void setNextPtr(Node *nextPtr);
```

NOTE: YOUR CONSTRUCTOR AND SETTERS DO NOT HAVE TO CHECK FOR VALID LATITUE, LONGITUDE OR DATES IN THIS HOMEWORK.

d. Your node class must implement the six equality/relation operators (i.e., ==, !=, <, >, <=, >=) based on m_day, m_month, and m_year. The class prototypes for these methods must be named exactly as followings.

```
bool operator==(const Node &rhs) const;
bool operator!=(const Node &rhs) const;
bool operator<(const Node &rhs) const;
bool operator>(const Node &rhs) const;
bool operator<=(const Node &rhs) const;
bool operator>=(const Node &rhs) const;
```

Hint: Try to minimize the amount of code that you need to rewrite by using operators that you have already coded. For example, realize that a != b is the same as !(a == b) and a >= b is the same as !(a < b).

2. **[50pts]** Implement a date-sorted, singly-linked list class called **MyList.**
   a. This class must have exactly two private member variables named **Node *m_startPtr and Node* m_currentPtr**. The m_startPtr pointer should always point to the starting node of the list. The m_currentPtr should point to the current node in the list.
   b. Your MyList class should define a default constructor (i.e., no inputs) that initializes the list to an empty list. In other words, both m_startPtr and m_currentPtr should be initialized to the nullptr.
   c. Your MyList must implement the following methods. The class prototypes for these methods must be named exactly as follows.

```
bool empty() const;
void print() const;
void insert(Node* node);
void remove(Node node);
void clear();
const Node * getCurrentNode();
void advanceToNextNode();
void resetCurrentPtr();
```

   d. The empty() method returns true if empty list, else false.
   e. The print() method prints the Nodes stored in the list. Print that the list is empty if the list is empty.
   f. The insert(Node * node) method inserts a new node into the list in date-sorted in order. You must dynamically create a new node and then pass this new node as the input to this method. This new node is inserted into the list so do not delete the node after calling the insert function. Hint: use the relational operators that you wrote for the Node class to locate where to insert the node into the list.
   g. The remove(Node node) method removes the first node from list that matches the date of the input node if it exists. Notice that the input is a Node object and not a pointer to a Node as was the case for insert. When removing the node from the list, remember to **delete** it (i.e., to ensure no memory leaks). If list is empty, print that the list was empty and there was nothing to delete. If the value was not found, print a message saying that number was not found in the list. Otherwise, print that the node was deleted. Hint: you will use the equality operators that you defined in the Node class.
   h. The clear() method removes all elements from the list. Print the value of each node that is deleted.
   i. Your MyList class should define a destructor that deletes all the nodes from the list. The destructor should print the value of each node that is deleted. Hint: use the clear() method.
   j. The getCurrentNode() method returns a pointer to the Node object at the m_currentPtr location. Note that this function returns a **const Node *** type so that you don't accidently modify the list outside of the class. Remember that a pointer of type **const Node *** is a pointer to constant Node data.
   k. The advanceToNextNode() method should move the m_currentPtr to the next node in the list. M_currentPtr must be set to nullptr if moves off the end of the list.
   l. The resetCurrentPtr method sets the m_currentPtr back to the start of the list.

3. **[25pts]** Add hard coded unit tests to the unitTests() function in main.cpp. Include descriptive print statements before each test or group of tests to make it easy to tell what you are testing.
   a. You should thoroughly test all the equality/relational operators in the Node classs. You do not need to test your getter and setter methods.
   b. You should thoroughly test all the methods in the MyList class. For example, insert nodes into an empty list, insert nodes at the beginning, middle and end of the list, delete nodes at the beginning, middle and end of the list, etc.

   Example unit tests:

```
Node n1(1,1,24,4,2020);
Node n2(1,1,25,4,2020);
cout << "Check if n1 < n2. Expected answer is 1. Computed answer is " << (n1 < n2) << endl;
cout << "Check if n1 == n2. Expected answer is 0. Computed answer is " << (n1 == n2) << endl;
cout << "Check if n1 <= n2. Expected answer is 1. Computed answer is " << (n1 <= n2) << endl;

MyList list;
cout << "Printing an empty list. Nothing should print." << endl;
list.print();
cout << "After printing and empty list." << endl;

cout << "insert 1, 2, 3, 4, 2020. Printing list." << endl;
Node * newNode = new Node(1, 2, 3, 4, 2020);
list.insert(newNode);
list.print();
```

Submission Instructions:

You must make a CLion project called "hw7" under your SVN homework directory. Check your homework into SVN.

Hint: you can see the current version of your submission by opening this link in a web browser:
https://class-svn.engineering.uiowa.edu/cie/projects/spring20/