COMP 53: Lists Lab, part 2

Instructions: In this lab, we are going to review singly-linked lists, focusing on insertion and removal of nodes.

- Get into groups of **at most two people** to accomplish this lab.
- At the top of your source code files list the group members as a comment.
- Each member of the group must individually submit the lab in Canvas.
- This lab includes **20 points** in aggregate. The details are given in the following.

1 `city.h` and `citynode.h`

Consider `city.h` and `citynode.h` as the one from the previous lab.

2 `citylist.h`

Consider `citylist.h` as the one from the previous lab, with two additional functions:

```
#ifndef CITYLIST_H
#define CITYLIST_H

#include<string>
#include "citynode.h"
class CityList {
        public:


                CityList() {
                        head = tail = nullptr;
                }

                void append(CityNode *cityNode) {

                }

                void prepend(CityNode *cityNode) {

                }

                void printCityList() {

                }

                CityNode *search(string cityName) {

                }

                void insert(CityNode *currNode, CityNode *cityNode) {

                }

                void remove(CityNode *currNode) {
```

```
        }


    private:
        CityNode *head;
        CityNode *tail;

};

#endif
```

Class `CityList` implements the singly-linked list of cities, which keeps track of the first and last elements of the list (through `head` and `tail` pointers, respectively). You have already completed the definition of functions `append(...)`, `prepend(...)`, `search(...)`, and `printCityList(...)`.

1. Complete the definition of `insert(...)` function that receives a pointer to a current `CityNode` along with a pointer to a new `CityNode`. It adds that node to the list as the successor of the current `CityNode`. *Hint*: You must consider special cases where the list is empty, as well as when the current node is the tail of the list ***(4 points)***.
2. Complete the definition of `remove(...)` function that receives a pointer to the current `CityNode`, and removes the node succeeding it. *Hint*: You must consider removing the list's head node as a special case ***(4 points)***.

3 main.cpp

In `main.cpp` do the following step by step:

1. Globally define array `cityArray[]` consisting of cities with the following details:
   (a) Los Angeles with population of 4340174
   (b) San Diego with population of 1591688
   (c) San Francisco with population of 871421
   (d) Sacramento with population of 505628
   (e) Stockton with the population of 323761
   (f) Redding with the population of 90292
   (g) Las Vegas with the population of 711926
   (h) Reno with the population of 289485
   (i) Portland with the population of 730428
   (j) Seattle with the population of 752180
   (k) Eugene with the population of 221452
2. Globally define a `CityList` named as `cityList` ***(1 points)***.
3. Pass `CityList` to these functions as *reference*.
   (a) Define function `void initCityListByAppend(...)` that receives a `CityList`, an array of elements of type `City` as a second input, and an integer as its third input. The third input represents the number of elements in the input array. Initialize the input `CityList` with the elements existing in the input array, by iteratively invoking `append()` function ***(3 points)***.

In `main()` function do the following step by step, using the functions defined above:

(i) Initialize `cityList` according to array `cityArray[]` by appending, using the function defined above ***(1 points)***.

(ii) Print out the entries of `cityList`, using the appropriate function defined as part of `CityList` class ***(1 points)***.

(iii) Add a `CityNode` for city Phoenix with population 1660472 after Stockton's node. Next, print out the resulting list. *Hint*: You can first search for Stockton, and use the pointer returned by the search function as the current node in insertion function ***(3 points)***.

(iv) Remove the node succeeding Reno's node. Next, print out the resulting list. *Hint*: You can first search for Reno, and use the pointer returned by the search function as the current node in removal function ***(3 points)***.

The output of the program may look like the following:

```
Initializing cityList with cityArray[] using appending:
Los Angeles: 4340174
San Diego: 1591688
San Francisco: 871421
Sacramento: 505628
Stockton: 323761
Redding: 90292
Las Vegas: 711926
Reno: 289485
Portland: 730428
Seattle: 752180
Eugene: 221452

Searching for Stockton in cityList, and inserting Phoenix after it:
Los Angeles: 4340174
San Diego: 1591688
San Francisco: 871421
Sacramento: 505628
Stockton: 323761
Phoenix: 1660472
Redding: 90292
Las Vegas: 711926
Reno: 289485
Portland: 730428
Seattle: 752180
Eugene: 221452

Searching for Reno in cityList, and removing the node after it:
Los Angeles: 4340174
San Diego: 1591688
San Francisco: 871421
Sacramento: 505628
Stockton: 323761
Phoenix: 1660472
Redding: 90292
Las Vegas: 711926
Reno: 289485
Seattle: 752180
Eugene: 221452
```