COMP 53: Lists Lab, part 6

Instructions: In this lab, we are going to review recursive traversal of lists, as well as implementation of stacks.

- 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 **34 points** in aggregate. The details are given in the following.

1 `city.h`

Consider `city.h` from the previous lab.

2 `citynode.h`

In `citynode.h` define class `CityNode` that implements doubly-linked nodes, i.e., they support two links: a link to the next node, and a link to the previous node. Define the default constructor accordingly ***(2 points)***.

Essentially a `CityNode` object is used as an element of the list for cities, which consists of a data component (a city), and a pointer to the next `CityNode`, and a pointer to the previous `CityNode`.

3 `citylist.h`

Consider class `CityList` as below (without dummy nodes):

```
#ifndef CITYLIST_H
#define CITYLIST_H

#include<string>
#include "citynode.h"
class CityList {
        public:
                CityNode *head;
                CityNode *tail;
                CityList() {
                        head = tail = nullptr;
                }
                void append(CityNode *cityNode);
                void prepend(CityNode *cityNode);
                void printCityList() {
                        printCityListRecursive(head);
                }
                CityNode *search(string cityName) {
                        return searchRecursive(head, cityName);
                }
                void remove(CityNode *currNode);
        private:
                void printCityListRecursive(CityNode *cityNode);
                CityNode *searchRecursive(CityNode *cityNode, string cityName);
};
#endif
```

1. Complete the definition of `void append(...)` function that receives a pointer to a `CityNode`, and adds that node to the end of the `CityList` ***(2 points)***.

2. Complete the definition of `void prepend(...)` function that receives a pointer to a `CityNode`, and adds that node to the beginning of the `CityList` ***(2 points)***.

3. Function `void printCityList()` function invokes function `void printCityListRecursive()`, a private function of the class. Complete the definition of this function. You must define it in a way that recursively the list is traversed and `printInfo()` is invoked on every node ***(4 points)***.

4. Function `CityNode *search(...)` function invokes function `CityNode *searchRecursive(...)`, a private function of the class. Complete the definition of this function. You must define it in a way that recursively the list is traversed to search for the city name, and return the pointer to the node with the matched name ***(4 points)***.

5. Complete the definition of `void remove(...)` function that receives a pointer to the current `CityNode`, and removes that node ***(2 points)***.

4 citystack.h

Consider `citystack.h` that defines a stack of cities as follows:

```
#ifndef CITYSTACK_H
#define CITYSTACK_H

#include "citylist.h"
class CityStack {
        public:
                CityStack(CityList &l) {
                        lst = l;
                }
                void pushCityNode(CityNode *cityNode);
                CityNode *popCityNode();
                CityNode *peekCityNode();
                bool isEmpty();
        private:
                CityList lst;
};

#endif
```

Complete the definition of functions

1. `void pushCityNode(...)` ***(2 points)***

2. `CityNode *popCityNode()` ***(2 points)***

3. `CityNode *peekCityNode()` ***(2 points)***

4. `bool isEmpty()` ***(2 points)***

5 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 ***(1 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) Search for node with the city name Stockton in `cityList`, and print out the population, if the search is successful ***(1 points)***.

(iv) Define a city stack `cityStack` and initialize it with `cityList` ***(1 points)***.

(v) Read the top of the stack and if not null, print out its name and population ***(1 points)***.

(vi) Push Phoenix with the population of 1660472 into `cityStack`, and then push Santa Fe with the population of 84263 ***(1 points)***.

(vii) Pop the top of the stack ***(1 points)***.

(viii) Read the top of the stack and if not null, print out its name and population ***(1 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 cityList1, if found print its population:
323761

Reading the top of cityStack:
Los Angeles: 4340174
Phoenix pushed.
Santa Fe pushed.
Top of stack is poped.
Reading the top of cityStack:
Phoenix: 1660472
```