COMP 53: Binary Search Tree Lab, part 2

Instructions: In this lab, we are going to review binary search trees (BSTs), specifically node removal, in-order traversal, and recursive search.

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

1 `city.h`

Consider `city.h` with the following details:

```
#ifndef CITY_H
#define CITY_H

#include<string>

class City {
        public:
                City() {
                        name = "N/A";
                        population = 0;
                }
                City(string nm, unsigned int pop) {
                        name = nm;
                        population = pop;
                }
                void setName(string name) {this -> name = name;}
                void setPopulation(unsigned int population)
                        {this -> population = population;}
                string getName() const {return this-> name;}
                unsigned int getPopulation() const {return this -> population;}
                virtual void printInfo() const {
                        cout<<getName()<<": "<<getPopulation()<<endl;
                }
        protected:
                string name;
                unsigned int population;
};

#endif
```

2 `citynode.h`

Consider `citynode.h` with the following details:

```
#ifndef CITYNODE_H
#define CITYNODE_H

#include<string>
#include "city.h"
```

```
class CityNode {
        public:
                City data;
                CityNode *left;
                CityNode *right;

                CityNode(City city) {
                        data = city;
                        left = nullptr;
                        right = nullptr;
                }
};
#endif
```

Essentially a `CityNode` object is used as a node of the BST for cities, which consists of a data component (a city), a pointer to the left subtree, and a pointer to the right subtree (both are pointers to `CityNode` objects).

3 citybst.h

Consider `citybst.h` with the following details:

```
#ifndef CITYBST_H
#define CITYBST_H

#include<string>
#include "citynode.h"
class CityBST {
        public:
                CityNode *root;

                CityBST() {
                        root = nullptr;
                }
                void insert(CityNode *cityNode);
                void printCityBST() {
                        printCityBSTRecursive(root,0);
                }
                void remove(unsigned int pop);
                CityNode *search(unsigned int pop);
                void printCityBST_InOrder() {
                        printCityBST_InOrderRecursive(root);
                }
                int getHeight() {
                        return getHeightRecursive(root);
                }
        private:
                void printCityBSTRecursive(CityNode *cityNode, int n);
                CityNode *searchRecursive(CityNode *cityNode, unsigned int pop);
                void printCityBST_InOrderRecursive(CityNode *cityNode);
                int getHeightRecursive(CityNode *cityNode);
};
```

```
#endif
```

Class `CityBST` implements the BST of cities, which keeps track of root `CityNode` of the BST (through `root`pointer).

1. Bring in the definition of `void insert(...)` and `void printCityBSTRecursive(...)` from the previous lab ***(4 points)***.
2. Complete the definition of `void remove(...)` that receives a population as input, looks it up the BST and removes the node associated with that population ***(6 points)***.
3. Function `CityNode *search(...)` invokes the private function `CityNode *searchRecursive(...)` that receives a city population. It is supposed to traverse the BST recursively and find the node with the associated population. This style of definition of different from the one in the previous lab, in which the function was defined non-recursively. Complete the definition of `CityNode *searchRecursive(...)` ***(4 points)***.
4. Function `void printCityBST_InOrder()` is used to define a second printing function for BSTs. It invokes the private function `void printCityBST_InOrderRecursive(...)` that is supposed to recursively traverse the BST, and calls `printInfo()` on each node's data component. Define `void printCityBST_InOrderRecursive(...)` in a way that handles traversal of the BST in **in-order** fashion. Note that `void printCityBSTRecursive(...)` does it in pre-order fashion. Another point is, the in-order printing of a BST generates a sorted sequence of the node keys ***(4 points)***.
5. Function `int getHeight()` invokes the private function `int getHeightRecursive(...)` that receives a `CityNode` as input. This function traverses the tree recursively and returns the height of the BST. Complete the defintion of `int getHeightRecursive(...)` ***(4 points)***.

4 main.cpp

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

1. Globally define array `cityArray[]` consisting of cities with the following details (in order):
   (a) Sacramento with population of 505628
   (b) Eugene with the population of 221452
   (c) Stockton with the population of 323761
   (d) Redding with the population of 90292
   (e) San Diego with population of 1591688
   (f) Reno with the population of 289485
   (g) Los Angeles with population of 4340174
   (h) Portland with the population of 730428
   (i) Las Vegas with the population of 711926
   (j) Seattle with the population of 752180
   (k) San Francisco with population of 871421
2. Globally define a `CityBST` named as `cityBST` ***(1 points)***.
3. Pass `CityBST` objects to the function below as *reference*.

(a) Define function `void initCityBSTByInsert(...)` that receives a `CityBST`, 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 `CityBST` with the elements existing in the input array, by iteratively invoking `insert()` function ***(2 points)***.

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

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

(ii) Print out the entries of `cityBST` in pre-order fashion, using the appropriate function defined as part of `CityBST` class ***(1 points)***.

(iii) Print out the height of `cityBST` ***(1 points)***.

(iv) Search for the city with population 289485 in `cityBST`, and if successful, read the name from the returned pointer to its node and print it in standard output. Otherwise, print that it is not found ***(1 points)***.

(v) Search for the city with population 782297 in `cityBST`, and if successful, read the name from the returned pointer to its node and print it in standard output. Otherwise, print that it is not found ***(1 points)***.

(vi) Remove the city with the population 871421 by invoking the appropriate function defined above. That city would be San Francisco. Next, print out the entries of `cityBST` in pre-order fashion ***(1 points)***.

(vii) Remove the city with the population 323761 by invoking the appropriate function defined above. That city would be Stockton. Next, print out the entries of `cityBST` in pre-order fashion ***(1 points)***.

(viii) Remove the city with the population 505628 by invoking the appropriate function defined above. That city would be Sacramento. Next, print out the entries of `cityBST` in pre-order fashion ***(1 points)***.

(ix) Print out the height of `cityBST` ***(1 points)***.

(x) Finally, traverse `cityBST` in in-order fashion and print out the city nodes ***(1 points)***.

The output of the program may look like the following:

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

Height of cityBST: 4

Searching in cityBST for the city with population 289485: Reno
Searching in cityBST for the city with population 782297: not found!

Removing city with population 871421 (San Francisco):
Sacramento: 505628
```

```
  Eugene: 221452
    Redding: 90292
    Stockton: 323761
      Reno: 289485
  San Diego: 1591688
    Portland: 730428
      Las Vegas: 711926
      Seattle: 752180
    Los Angeles: 4340174

Removing city with population 323761 (Stockton):
Sacramento: 505628
  Eugene: 221452
    Redding: 90292
    Reno: 289485
  San Diego: 1591688
    Portland: 730428
      Las Vegas: 711926
      Seattle: 752180
    Los Angeles: 4340174

Removing city with population 505628 (Sacramento):
Las Vegas: 711926
  Eugene: 221452
    Redding: 90292
    Reno: 289485
  San Diego: 1591688
    Portland: 730428
      Seattle: 752180
    Los Angeles: 4340174

Height of cityBST: 3

cityBST inorder traversal and printing:
Redding: 90292
Eugene: 221452
Reno: 289485
Las Vegas: 711926
Portland: 730428
Seattle: 752180
San Diego: 1591688
Los Angeles: 4340174
```