# COMP 53: Binary Search Tree Lab, part 1

***Instructions:*** In this lab, we are going to review binary search trees (BSTs).

- 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 **23 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);
                CityNode *search(unsigned int pop);
                void printCityBST() {
                        printCityBSTRecursive(root,0);
                }
        private:
                void printCityBSTRecursive(CityNode *cityNode, int n);

};

#endif
```

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

1. Complete the definition of `void insert(...)` function that receives a pointer to a `CityNode`, and adds that node to the BST. You need to insert the node into the tree according to the city's *population* ***(5 points)***.

2. Complete the definition of `CityNode *search(...)` function that receives a city population (an unsigned integer). It traverses the BST to find the city with that population, and returns a pointer to that node if successful. Otherwise, it returns null pointer ***(5 points)***.

3. Function `printCityBST()` invokes the private function `void printCityBSTRecursive(...)` that is supposed to recursively traverse the BST, and calls `printInfo()` on each node's data component. This function receives a pointer to the current `CityNode`, along with an integer that represents the number of indentations that is needed to print that `CityNode`. Follow these step, in order, to complete the definition of `void printCityBSTRecursive(CityNode *cityNode, int n)`:

   (a) Check if the input `cityNode` is null. If so, return. (There is nothing to print!)

   (b) Print white space for $n$ times.

   (c) Print the information of the city pointed by `cityNode`. (Call `printInfo()`!)

   (d) Recursively call the function on the *left* subtree of `cityNode` with indentation number `n+1`.

   (e) Recursively call the function on the *right* subtree of `cityNode` with indentation number `n+1`.

   This style of traversal of the binary tree is called **preorder traversal** ***(5 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` objewcts 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 ***(3 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`, using the appropriate function defined as part of `CityBST` class ***(1 points)***.

(iii) 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)***.

(iv) 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)***.

The output of the program may look like the following:

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

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