COMP 53: Hash Tables Lab, part 2

Instructions: In this lab, we are going to review hash tables that use linear probing for collisions.

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

1 `city.h`

Consider `city.h` from the previous lab.

```
#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 `citybucket.h`

Class `CityBucket` implements city buckets for linear probing. Each bucket consists of a city along with a state. The state represents whether the bucket is empty since start, empty after removal, or filled with a city. Enumerated type `BucketState` is defined for this purpose.

```
#ifndef CITYBUCKET_H
#define CITYBUCKET_H

#include<string>
```

```
#include "city.h"
enum BucketState {empty_since_start, empty_after_removal, filled};

class CityBucket {
        public:
                City city;
                BucketState state;

                CityBucket() {
                        city = City();
                        state = empty_since_start;
                }
};
#endif
```

3 cityhash.h

This file includes the definition of class `CityHashTable` that implements hash tables for cities. It handles collision by linear probing (i.e., `CityBucket` in each bucket). Note that we use city population as the *key* for hash table entries.

```
#ifndef CITYHASH_H
#define CITYHASH_H

#include <string.h>
#include "citybucket.h"

const int maxArraySize = 100;

class CityHashTable {
        public:
                CityHashTable() {
                        tableSize = 0;
                }
                CityHashTable(int size) {
                        tableSize = size;
                }
                CityBucket *search(unsigned int pop);
                bool insert(City inputCity);
                void remove(City city);
                void printHashTable();
        private:
                CityBucket table[maxArraySize];
                int tableSize;
                int hash(unsigned int pop);
};
#endif
```

A `CityHashTable` includes an array of `CityBucket`s, called `table`. The size of `table` is kept in `tableSize`.

1. Complete the definition of the function `int hash(...)` that receives a population and returns the bucket number in the hash table. Use modulo `tableSize` for this purpose ***(2 points)***.

2. Complete the definition of the function `CityBucket *search(...)` that receives population (as the key) and returns the address of the `CityBucket` that points to the city with that population within the hash table. If not found, it returns null pointer. *Hint*: You need to traverse the table starting with the appropriated bucket index ***(3 points)***.

3. Complete the definition of the function `bool insert(...)` that receives a city and adds it to the hash table. In this case, it returns `true`. If insertion is unsuccessful (when table is full), then it returns `false`. *Hint*: You need to traverse the table starting with the appropriated bucket index ***(3 points)***.

4. Complete the definition of the function `void remove(...)` that receives a city. It searches for the city in the hash table and if found deletes it from the hash table. *Hint*: You need to traverse the table starting with the appropriated bucket index. Do not forget to update the state of the bucket upon removal ***(3 points)***.

5. Complete the definition of the function `void printHashTable()` that traverses the buckets and prints the state and item in the hash table. *Hint*: You need to traverse the table starting with the appropriated bucket index ***(3 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:
   (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 two `CityHashTables` named as `cityHT1` and `cityHT2` ***(1 points)***.

3. Pass `CityHashTables` to the function below as *reference*.
   (a) Define function `void initCityHT(...)` that receives a `CityHashTable`, 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 `CityHashTable` 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 `cityHT1` with 15 buckets and entries coming from `cityArray[]` ***(1 points)***.

(ii) Print out the entries of `cityHT1`, using the appropriate function defined as part of `CityHashTable` class ***(1 points)***.

(iii) Initialize `cityHT2` with 10 buckets and entries coming from `cityArray[]` ***(1 points)***.

(iv) Print out the entries of `cityHT2`, using the appropriate function defined as part of `CityHashTable` class ***(1 points)***.

(v) Search for the city name with population 752180 in `cityHT2` ***(1 points)***.

(vi) Remove Seattle from `cityHT2`, and print out the updated hash table ***(1 points)***.

(vii) Insert Phoenix with population 1660472 into `cityHT2`, and print out the updated hash table ***(1 points)***.

The output of the program may look like the following:

```
Initializing cityHT1 with 15 buckets and entries coming from cityArray[]:
--Bucket 0 filled Reno: 289485
--Bucket 1 filled Stockton: 323761
--Bucket 2 empty_since_start N/A: 0
--Bucket 3 filled Portland: 730428
--Bucket 4 empty_since_start N/A: 0
--Bucket 5 filled Seattle: 752180
--Bucket 6 empty_since_start N/A: 0
--Bucket 7 filled Eugene: 221452
--Bucket 8 filled Sacramento: 505628
--Bucket 9 filled Redding: 90292
--Bucket 10 filled San Diego: 1591688
--Bucket 11 filled Las Vegas: 711926
--Bucket 12 filled San Francisco: 871421
--Bucket 13 empty_since_start N/A: 0
--Bucket 14 filled Los Angeles: 4340174

Initializing cityHT2 with 10 buckets and entries coming from cityArray[]:
--Bucket 0 filled Portland: 730428
--Bucket 1 filled Stockton: 323761
--Bucket 2 filled Eugene: 221452
--Bucket 3 filled Redding: 90292
--Bucket 4 filled Los Angeles: 4340174
--Bucket 5 filled Reno: 289485
--Bucket 6 filled Las Vegas: 711926
--Bucket 7 filled Seattle: 752180
--Bucket 8 filled Sacramento: 505628
--Bucket 9 filled San Diego: 1591688

Searching for the city name with population 752180 in cityHT2: Seattle

Removing Seattle from cityHT2:
--Bucket 0 filled Portland: 730428
--Bucket 1 filled Stockton: 323761
--Bucket 2 filled Eugene: 221452
--Bucket 3 filled Redding: 90292
--Bucket 4 filled Los Angeles: 4340174
--Bucket 5 filled Reno: 289485
--Bucket 6 filled Las Vegas: 711926
--Bucket 7 empty_after_removal N/A: 0
--Bucket 8 filled Sacramento: 505628
--Bucket 9 filled San Diego: 1591688
```

```
Inserting Phoenix with population 1660472 into CityHT2:
--Bucket 0 filled Portland: 730428
--Bucket 1 filled Stockton: 323761
--Bucket 2 filled Eugene: 221452
--Bucket 3 filled Redding: 90292
--Bucket 4 filled Los Angeles: 4340174
--Bucket 5 filled Reno: 289485
--Bucket 6 filled Las Vegas: 711926
--Bucket 7 filled Phoenix: 1660472
--Bucket 8 filled Sacramento: 505628
--Bucket 9 filled San Diego: 1591688
```