COMP 53: Hash tables Lab, part 1

Instructions: In this lab, we are going to review hash tables that use chaining 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 **33 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 citynode.h

`citynode.h` defines `CityNode`s with double links. 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`.

```
#ifndef CITYNODE_H
#define CITYNODE_H

#include "city.h"
```

```
class CityNode {
        public:
                City data;
                CityNode *next;
                CityNode *prev;
                CityNode(City city) {
                        data = city;
                        next = nullptr;
                        prev = nullptr;
                }
};
#endif
```

3 citylist.h

Class `CityList` implements the doubly-linked list of cities, which keeps track of the first and last elements of the list (through `head` and `tail` pointers, respectively).

```
#ifndef CITYLIST_H
#define CITYLIST_H

#include<string>
#include "citynode.h"
class CityList {
        public:
                CityList() {
                        head = tail = nullptr;
                }
                void append(CityNode *cityNode);
                void printCityList();
                CityNode *search(unsigned  int pop);
                void remove(CityNode *currNode);
        private:
                CityNode *head;
                CityNode *tail;
};
#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 printCityList()` function that traverses through the elements of the `CityList`, and calls `printInfo()` on each node's data component ***(2 points)***.

3. Complete the definition of `CityNode *search(...)` function that receives a city population. It traverses through the elements of the `CityList` to find the city with that name, and returns a pointer to that node if successful. Otherwise, it returns null pointer ***(2 points)***.

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

You have defined the aforementioned functions before.

4 cityhash.h

This file includes the definition of class `CityHashTable` that implements hash tables for cities. It handles collision by chaining (i.e., `CityList` 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 "citylist.h"

const int maxArraySize = 100;

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

A `CityHashTable` includes an array of `CityList`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 `CityNode *search(...)` that receives population (as the key) and returns the address of the `CityNode` that points to the city with that population within the hash table. If not found, it returns null pointer. *Hint*: Use `search()` from city lists ***(3 points)***.

3. Complete the definition of the function `void insert(...)` that receives a city and adds it to the hash table. *Hint*: Use `append()` function from city lists ***(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*: Use `remove()` function from city lists ***(3 points)***.

5. Complete the definition of the function `void printHashTable()` that traverses the buckets and prints each item in the hash table. *Hint*: Use `printCityList()` function from city lists ***(3 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) 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 10 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 5 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 1591688 in cityHT2 ***(1 points)***.

(vi) Remove San Diego 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 10 buckets and entries coming from cityArray[]:
--Bucket 0:
Seattle: 752180
--Bucket 1:
Stockton: 323761
San Francisco: 871421
```

```
--Bucket 2:
Eugene: 221452
Redding: 90292
--Bucket 3:
--Bucket 4:
Los Angeles: 4340174
--Bucket 5:
Reno: 289485
--Bucket 6:
Las Vegas: 711926
--Bucket 7:
--Bucket 8:
Sacramento: 505628
San Diego: 1591688
Portland: 730428
--Bucket 9:

Initializing cityHT2 with 5 buckets and entries coming from cityArray[]:
--Bucket 0:
Reno: 289485
Seattle: 752180
--Bucket 1:
Stockton: 323761
Las Vegas: 711926
San Francisco: 871421
--Bucket 2:
Eugene: 221452
Redding: 90292
--Bucket 3:
Sacramento: 505628
San Diego: 1591688
Portland: 730428
--Bucket 4:
Los Angeles: 4340174

Searching for the city name with population 1591688 in cityHT2: San Diego

Removing San Diego from cityHT2:
--Bucket 0:
Reno: 289485
Seattle: 752180
--Bucket 1:
Stockton: 323761
Las Vegas: 711926
San Francisco: 871421
--Bucket 2:
Eugene: 221452
Redding: 90292
--Bucket 3:
Sacramento: 505628
Portland: 730428
--Bucket 4:
Los Angeles: 4340174
```

```
Inserting Phoenix with population 1660472 into CityHT2:
--Bucket 0:
Reno: 289485
Seattle: 752180
--Bucket 1:
Stockton: 323761
Las Vegas: 711926
San Francisco: 871421
--Bucket 2:
Eugene: 221452
Redding: 90292
Phoenix: 1660472
--Bucket 3:
Sacramento: 505628
Portland: 730428
--Bucket 4:
Los Angeles: 4340174
```