COMP 53: Containers Lab, part 5

Instructions: In this lab, we are going to review queues, deques, and sorting.

- 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`

Use `city.h` from the previous lab without any modifications.

2 `main.cpp`

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

1. Include `queue`, `deque`, `vector`, and `algorithm` as well as `city.h`.
2. Globally define array `cityArray[]` consisting of cities with the following details:
   (a) Los Angeles with population of 4 million
   (b) San Diego with population of 1.5 million
   (c) San Francisco with population of 900 thousand
   (d) Sacramento with population of 500 thousand
   (e) Stockton with the population of 300 thousand
3. Globally define a queue of `City` objects, without initial values. Call it `cityQueue` ***(1 points)***.
4. Globally define a deque of `City` objects, without initial values. Call it `cityStack` ***(1 points)***.
5. Globally define a vector of `City` objects, without initial values. Call it `cityVector` ***(1 points)***.
6. Define the following functions on queues, deques and vectors. Pass all containers to these functions as *reference*.
   (a) Define function `void initQueue(...)` that receives a queue of `City` objects, 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 queue with the elements existing in the input array ***(3 points)***.
   (b) Define function `void printCityQueue(...)` that receives a queue of `City` objects as input and prints the elements within the queue. *Hint*: Note that you cannot traverse the elements of a queue (in contrast to already studied containers). In order to define this function you can iteratively print the front element of the input queue, remove it from the input queue, and then add it the end of that queue ***(4 points)***.
   (c) Define function `void initStack(...)` that receives a deque of `City` objects, 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 deque with the elements existing in the input array. Note that you must treat the deque as a stack, i.e., you cannot use `push_back()`, `pop_back()` and `back()` ***(3 points)***.

(d) Define function `void printCityStack(...)` that receives a deque of `City` objects as input and prints the elements within the deque. Note that you must treat the deque as a stack, i.e., you cannot use `push_back()`, `pop_back()` and `back()`. *Hint*: Similar to queue, you cannot traverse the elements of a stack (in contrast to already studied containers). In order to define this function:

- define a second stack in the body of the function
- iteratively print the top element of the input stack, pop it from stack, and push it into the second stack.
- iteratively pop elements from the second stack and push them back to the first stack.

This way, you can print all of the elements of the input stack, while maintaining its structure ***(5 points)***.

(e) Define function `void initVector(...)` that receives a vector of `City` objects, 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 queue with the elements existing in the input array ***(2 points)***.

(f) Define function `void printCityVector(...)` that receives a vector of `City` objects as input and prints the elements within the vector. *Hint*: You can use range-based `for` loops ***(2 points)***.

(g) Define a function that receives two cities as input and returns true if name of the first city is larger than name of the second city. Otherwise it returns false ***(2 points)***.

(h) Define a function that receives two cities as input and returns true if population of the first city is less than population of the second city. Otherwise it returns false ***(2 points)***.

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

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

(ii) Print out the entries of `cityQueue`, using the appropriate function defined above ***(1 points)***.

(iii) Initialize `cityStack` according to array `cityArray[]` using the function defined above ***(1 points)***.

(iv) Print out the entries of `cityStack`, using the appropriate function defined above ***(1 points)***.

(v) Initialize `cityVector` according to array `cityArray[]` using the function defined above ***(1 points)***.

(vi) Print out the entries of `cityVector`, using the appropriate function defined above ***(1 points)***.

(vii) Sort `cityVector` based on names in descending order, and print out the updated vector. *Hint*: Use `sort()` function ***(1 points)***.

(viii) Sort `cityVector` based on populations in ascending order, and print out the updated vector. *Hint*: Use `sort()` function ***(1 points)***.

The output of the program may look like the following:

```
Initializing cityQueue with cityArray[]:
Los Angeles: 4000000
San Diego: 1500000
San Francisco: 900000
Sacramento: 500000
Stockton: 300000

Initializing cityStack with cityArray[]:
Stockton: 300000
```

```
Sacramento: 500000
San Francisco: 900000
San Diego: 1500000
Los Angeles: 4000000

Initializing cityVector with cityArray[]:
Los Angeles: 4000000
San Diego: 1500000
San Francisco: 900000
Sacramento: 500000
Stockton: 300000


Sorting cityVector based on names in descending order:
Stockton: 300000
San Francisco: 900000
San Diego: 1500000
Sacramento: 500000
Los Angeles: 4000000


Sorting cityVector based on populations in ascending order:
Stockton: 300000
Sacramento: 500000
San Francisco: 900000
San Diego: 1500000
Los Angeles: 4000000
```