COMP 53: Containers Lab, part 4

Instructions: In this lab, we are going to review sets.

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

1 `city.h`

Use `city.h` from the previous lab with the following addition.

1. Overload operator < for `City` by comparing the names of `City` object referred by `this` and an input `City` object. You may start its definition by `bool operator<(const City &city) const {...}`

   We need to define this operator for `City` so that we make sets of such objects later on ***(2 points)***.

2 `main.cpp`

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

1. Include `set` and `utility` as well as `city.h`.
2. Globally define array `a[]` of integers consisting of values: 2,8,1,7, and 3 in order.
3. Globally define array `cityArray1[]` 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
4. Globally define array `cityArray2[]` consisting of cities with the following details:
   (a) Redding with population of 90 thousand
   (b) Stockton with the population of 300 thousand
   (c) Las Vegas with the population of 700 thousand
   (d) Reno with population of 300 thousand
   (e) Portland with population of 700 thousand
   (f) Sacramento with population of 500 thousand
   (g) Eugene with population of 200 thousand
5. Globally define a set of integers, without initial values. Call it `intSet` ***(1 points)***.
6. Globally define three sets of cities, without initial values. Call them `citySet1`, `citySet2`, and `citySet3` ***(1 points)***.
7. Globally define a set of strings, without initial values. Call it `cityNames` ***(1 points)***.

8. Define the following functions on sets. Pass all sets to these functions as *reference*. Moreover, annotate the input set to be *constant*, if the function is not supposed to modify that input set. In addition, use *range-based `for` loops* to traverse the sets.

   (a) Define function `void initSet(...)` that receives a set of elements of *any arbitrary type* T, an array of elements of type T 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 set with the elements existing in the input array ***(3 points)***.

   (b) Define function `void printSet(...)` that receives a set of elements of *any arbitrary type* T as input and prints the elements residing within that set in the standard output ***(3 points)***.

   (c) Define a specific printing function for set of cities. Specifically, define function `void printCitySet(...)` that receives a set of cities as input and prints the elements residing within that set in the standard output (using `printInfo()` function) ***(3 points)***.

   (d) Define function `void cityNamesSet(...)` that receives a set of cities, along with a set of strings. It traverses the set of cities and inserts the city names to the input set of strings ***(3 points)***.

   (e) Define function `void setIntersection(...)` that receives three sets as input, where all elements have the same *arbitrary type* T. It inserts the intersection of the first two sets into the third set ***(3 points)***.

   (f) Define function `void setUnion(...)` that receives three sets as input, where all elements have the same *arbitrary type* T. It inserts the union of the first two sets into the third set ***(3 points)***.

   (g) Define function `bool isSubset(...)` that receives two sets as input, where all elements have the same *arbitrary type* T. It returns `true` if the first input set is a subset of second input set. Otherwise it returns `false` ***(3 points)***.

   (h) Define function `void removeFromCitySet(...)` that receives a set of cities as input, along with a name of city (a string). It removes the city matched with the input city name from the input set ***(3 points)***.

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

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

(ii) Print out the elements of `intSet`, using to the appropriate function defined above ***(1 points)***.

(iii) Initialize `citySet1` according to array `cityArray1[]` using the function defined above ***(1 points)***.

(iv) Print out the elements of `citySet1`, using to the appropriate function defined above ***(1 points)***.

(v) Initialize `citySet2` according to array `cityArray2[]` using the function defined above ***(1 points)***.

(vi) Print out the elements of `citySet2`, using to the appropriate function defined above ***(1 points)***.

(vii) Populate `cityNames` according to the cities in `citySet1` by invoking the appropriate function defined above ***(1 points)***.

(viii) Print out the elements of `cityNames`, using to the appropriate function defined above ***(1 points)***.

(ix) Insert the intersection of `citySet1` and `citySet2` into `citySet3`, using the function defined above. Then, print out the elements of `citySet3` ***(1 points)***.

(x) Insert the union of `citySet1` and `citySet2` into `citySet3`, using the function defined above. Then, print out the elements of `citySet3` ***(1 points)***.

(xi) Check whether `citySet1` is a subset of `citySet2` ***(1 points)***.

(xii) Check whether `citySet2` is a subset of `citySet3` ***(1 points)***.

(xiii) Remove city named Sacramento from `citySet1` using the function defined above. Next, print out the updated set ***(1 points)***.

The output of the program may look like the following:

```
Initializing intSet with a[]:
1 2 3 7 8

Initializing citySet1 with cityArray1[]:
Los Angeles: 4000000
Sacramento: 500000
San Diego: 1500000
San Francisco: 900000
Stockton: 300000

Initializing citySet2 with cityArray2[]:
Eugene: 200000
Las Vegas: 700000
Portland: 700000
Redding: 90000
Reno: 300000
Sacramento: 500000
Stockton: 300000

Populate cityNames according to the cities in citySet1:
Los Angeles Sacramento San Diego San Francisco Stockton

Intersect citySet1 and citySet2, and store the result in citySet3:
Sacramento: 500000
Stockton: 300000

Union citySet1 and citySet2, and store the result in citySet3:
Eugene: 200000
Las Vegas: 700000
Los Angeles: 4000000
Portland: 700000
Redding: 90000
Reno: 300000
Sacramento: 500000
San Diego: 1500000
San Francisco: 900000
Stockton: 300000

citySet1 is a subset of citySet2? 0
citySet2 is a subset of citySet3? 1

Removing Sacramento from citySet1:
Los Angeles: 4000000
San Diego: 1500000
San Francisco: 900000
Stockton: 300000
```