National Research University Higher School of Economics
Faculty of Computer Science
Bachelor’s Program “HSE University and University of London Double Degree
Program in Data Science and Business Analytics”

Introduction to Programming

Workshop #15

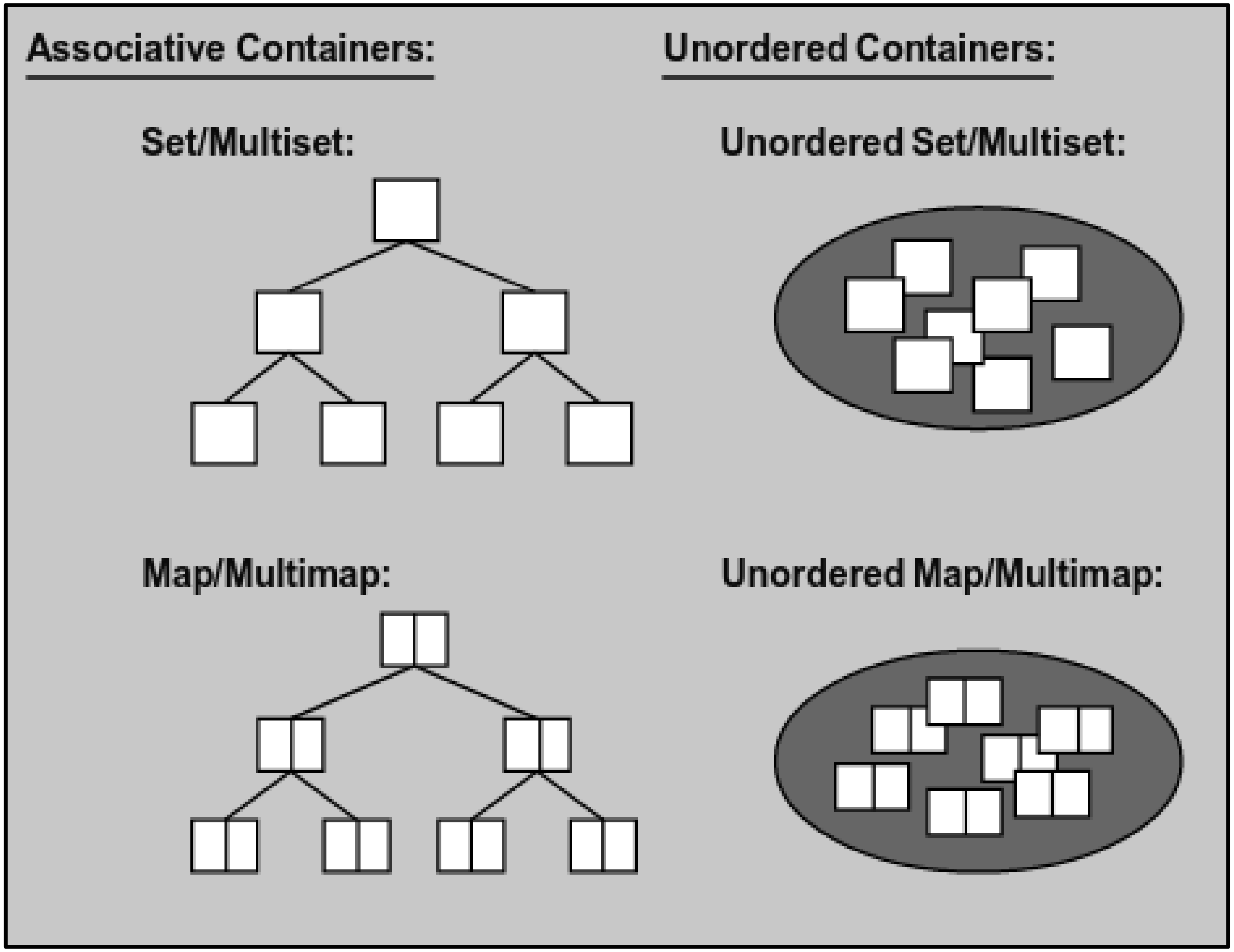
Wed 03.03.2021

Julio Carrasquel

NATIONAL RESEARCH
UNIVERSITY

Our topic in previous workshops...

`std :: [unordered_] [multi] maps and sets`



Containers such as `std::vector`, `std::map` **or** `std::set`
can be composed *not only* by “atomic” data types like...

```
std::vector<std::string>

std::vector<int>

std::map<std::string, int>

std::set<int>

std::map<int, bool>

...
```

Depending on the problem to solve, we can have containers with *complex* structures...

```
std::map<std::string, std::pair<double,double>>
```

Depending on the problem to solve, we can have containers with *complex* structures...

```
std::map<std::string, std::pair<double,double>>
```

```
Example: a map where keys are names of cities, and values are their
coordinates.
cityMap[“New York”] = (40.6943, -73.9249)
cityMap[“Shanghai”] = (31.1667, 121.4667)
…
```

Depending on the problem to solve, we can have containers with *complex* structures...

```
std::map<std::string, std::pair<double,double>>
```

```
Example: a map where keys are names of cities, and values are their
coordinates.
cityMap[“New York”] = (40.6943, -73.9249)
cityMap[“Shanghai”] = (31.1667, 121.4667)
…
```

```
std::map<double,std::pair<std::string, std::string>>
```

```
Example: a map to store pairs of cities and their distances.
distanceMap[11860.47] = (“New York”, “Shanghai”)
…
```

Depending on the problem to solve, we can have containers with *complex* structures...

```
std::map<std::string, std::pair<double,double>>
```

```
Example: a map where keys are names of cities, and values are their
coordinates.
cityMap[“New York”] = (40.6943, -73.9249)
cityMap[“Shanghai”] = (31.1667, 121.4667)
…
```

```
std::map<double,std::pair<std::string, std::string>>
```

```
Example: a map to store pairs of cities and their distances.
distanceMap[11860.47] = (“New York”, “Shanghai”)
…
```

Depending on the problem to solve, we can have containers with *complex* structures...

```
std::map<std::string, std::pair<double,double>>
```

```
Example: a map where keys are names of cities, and values are their
coordinates.
cityMap[“New York”] = (40.6943, -73.9249)
cityMap[“Shanghai”] = (31.1667, 121.4667)
…
```

```
std::map<double,std::pair<std::string, std::string>>
```

```
Example: a map to store pairs of cities and their distances.
distanceMap[11860.47] = (“New York”, “Shanghai”)
…
```

For *this problem*, we also could have declared the container as follows:

```
std::map<std::pair<std::string, std::string>, double>
```

```
Keys are pairs between cities, and values give their distance.
distanceMap[(“New York”, “Shanghai”)] = 11860.47
```

BUT, in this problem we wanted to sort the map by the distance between cities, and this is why we choose double as the key.

Depending on the problem to solve, we can have containers with *complex* structures...

```
std::map<std::string, std::pair<double,double>>
```

```
Example: a map where keys are names of cities, and values are their
coordinates.
cityMap[“New York”] = (40.6943, -73.9249)
cityMap[“Shanghai”] = (31.1667, 121.4667)
…
```

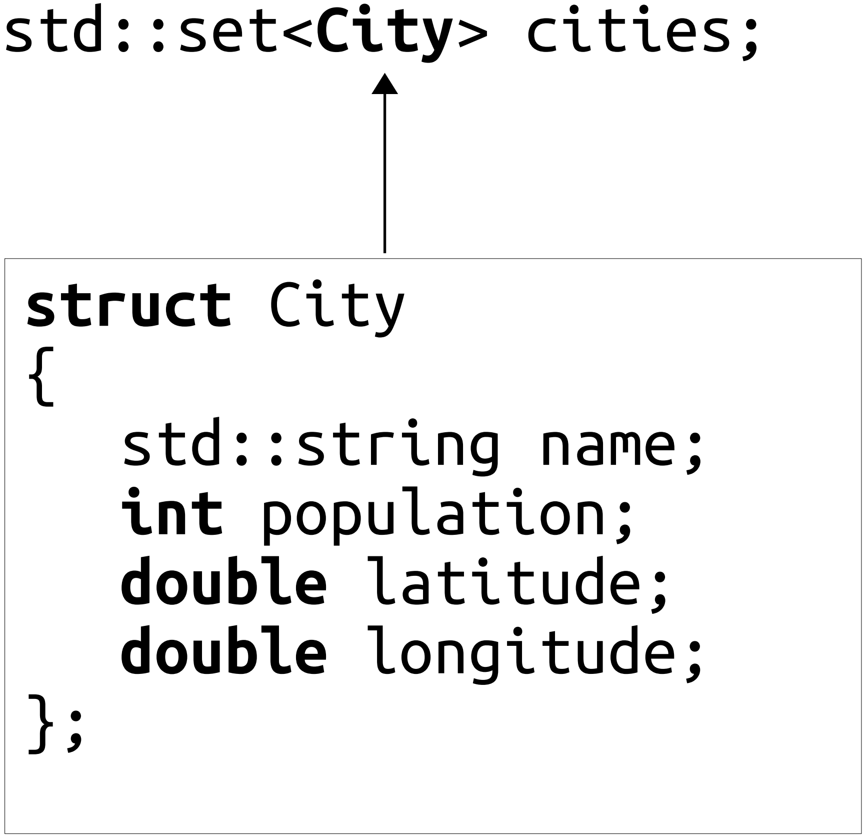
```
std::map<double,std::pair<std::string, std::string>>
```

```
Example: a map to store pairs of cities and their distances.
distanceMap[11860.47] = (“New York”, “Shanghai”)
…
```

```
std::map<std::string, std::vector<std::pair<std::string,int>>
```

```
Example: a map where keys are country names, and values are vectors of pairs city-population.

countryMap[“USA”] = [ (“New York”,18713220) , (“Los Angeles”,12750807) ]
countryMap[“China”] = [ (“Shanghai”,22120000) , (“Guangzhou”,20902000) ) , (“Beijing”,18713220) ]
```

When resolving problems with `structures` and `classes`, we also can use containers…



```
std::set<City> cities;
```

```
struct City
{
   std::string name;
   int population;
   double latitude;
   double longitude;
};
```

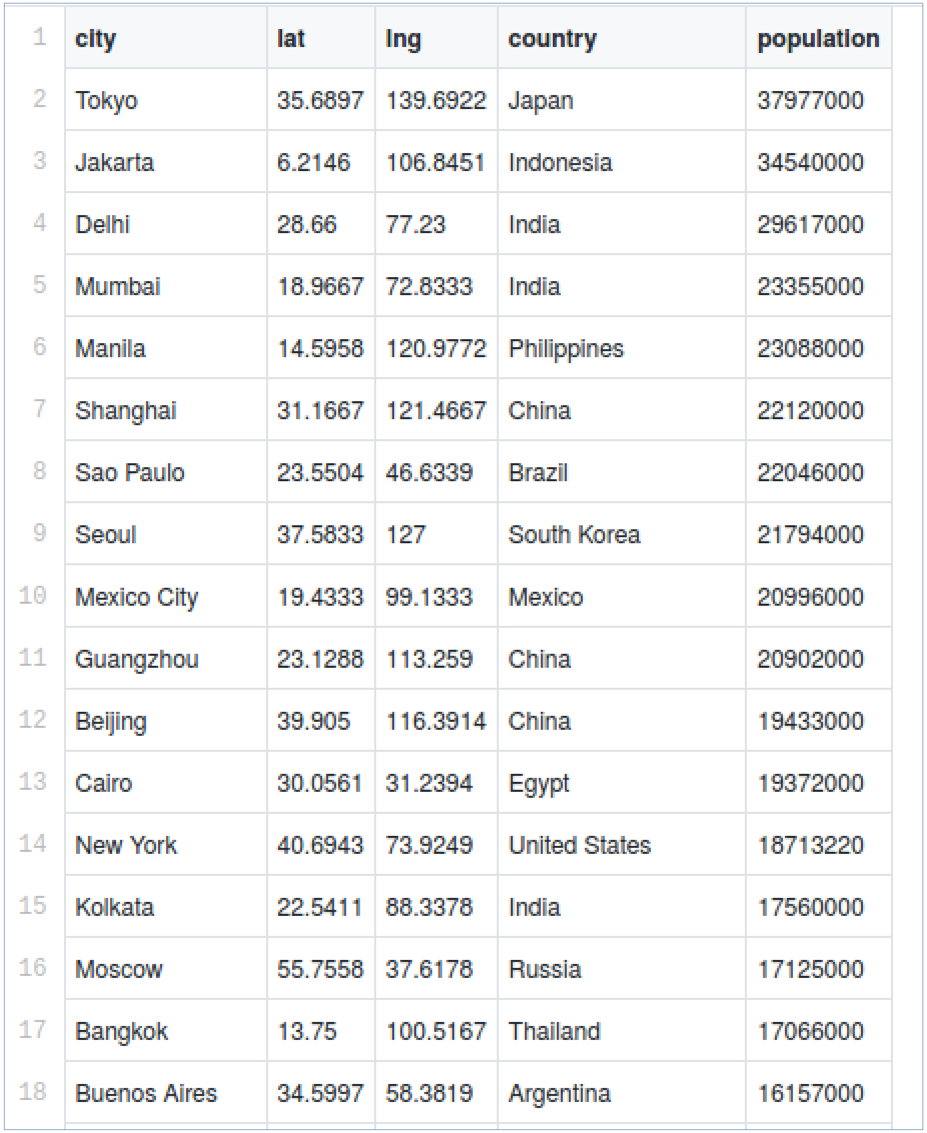
When resolving problems with `structures` and `classes`, we also can use containers…

```
std::vector<FootballTeam> teams;
```

```
struct FootballTeam
{
   std::string name;
   std::string city;
   std::string stadium;
   int points;
};
```

Previous seminar’s problem – `cities.csv`

File with the 500 most populated cities in the world



city	lat	lng	country	population
Tokyo	35.6897	139.6922	Japan	37977000
Jakarta	6.2146	106.8451	Indonesia	34540000
Delhi	28.66	77.23	India	29617000
Mumbai	18.9667	72.8333	India	23355000
Manila	14.5958	120.9772	Philippines	23088000
Shanghai	31.1667	121.4667	China	22120000
Sao Paulo	23.5504	46.6339	Brazil	22046000
Seoul	37.5833	127	South Korea	21794000
Mexico City	19.4333	99.1333	Mexico	20996000
Guangzhou	23.1288	113.259	China	20902000
Beijing	39.905	116.3914	China	19433000
Cairo	30.0561	31.2394	Egypt	19372000
New York	40.6943	73.9249	United States	18713220
Kolkata	22.5411	88.3378	India	17560000
Moscow	55.7558	37.6178	Russia	17125000
Bangkok	13.75	100.5167	Thailand	17066000
Buenos Aires	34.5997	58.3819	Argentina	16157000

Previous seminar’s problem – Task 1

1) Create a `std::map<std::string, std::vector<std::pair<std::string,int> >` called `countryMap` where the keys are *country names*, and the values are vectors of *pairs city-population*. Fill the map with the information in the file `cities.csv`

Example of countryMap

```
countryMap[“USA”] = [ (“New York”,18713220) , (“Los Angeles”,12750807) ]

countryMap[“China”] = [ (“Shanghai”,22120000) , (“Guangzhou”,20902000) ) , (“Beijing”,18713220) ]

...
```

2) Print the vector of cities-population of the 5 *countries with most population.* To resolve this task, use the first map `countryMap` to create a second map `std::map<int , std::string> populationMap` where the keys are the *sum of city population of a country* and the values are the *country names*.

Example of populationMap

```
populationMap[31464027] = “USA”

populationMap[61735220] = “China”

...
```

Then, for finding the 5 *countries with most population*, we need to take the *country names* from *the last 5 elements* of `populationMap` (why? remember, the map is ordered)
Finally, print the list of vectors city-population of the 5 countries you took.

Previous seminar's problem – Task 2

1) Create a `std::map<std::string, std::pair<double,double>>` called `cityMap` where the keys are *city names*, and the values are pairs *latitude-longitude.*
Fill the map with the information in the file `cities.csv`

Example of cityMap

```
cityMap["New York"] = (40.6943, -73.9249)

cityMap["Shanghai"] = (31.1667, 121.4667)

...
```

2) Print the 5 pairs of cities with the *farthest distances* between each other.
To resolve this task, use the first map `cityMap` to create a second map
`std::map<double , std::pair<std::string, std::string>> distanceMap`
where keys are distances between cities* and values are pairs of cities.

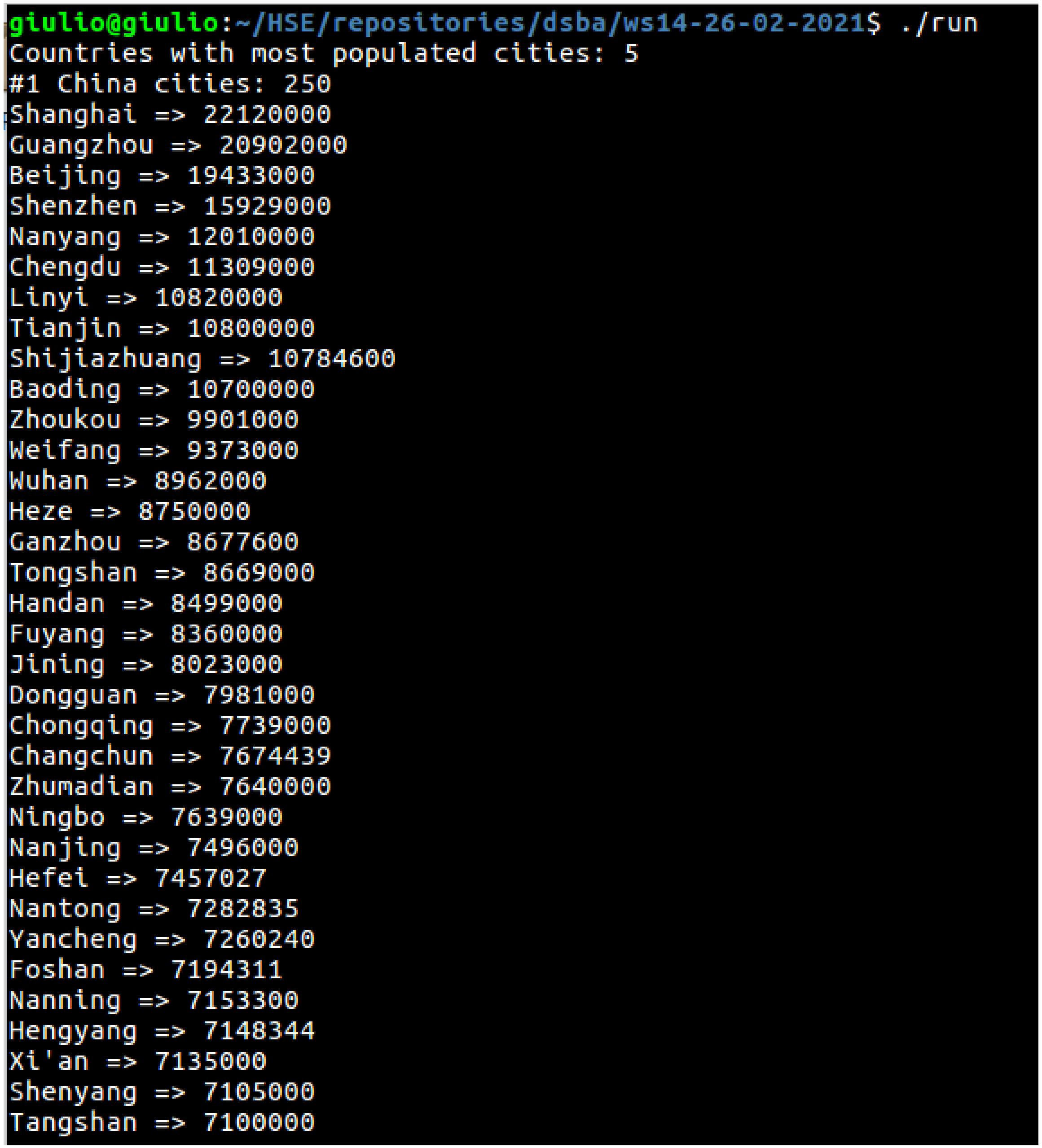
Example of distanceMap

```
distanceMap[11860.47] = ("New York", "Shanghai")
...
```

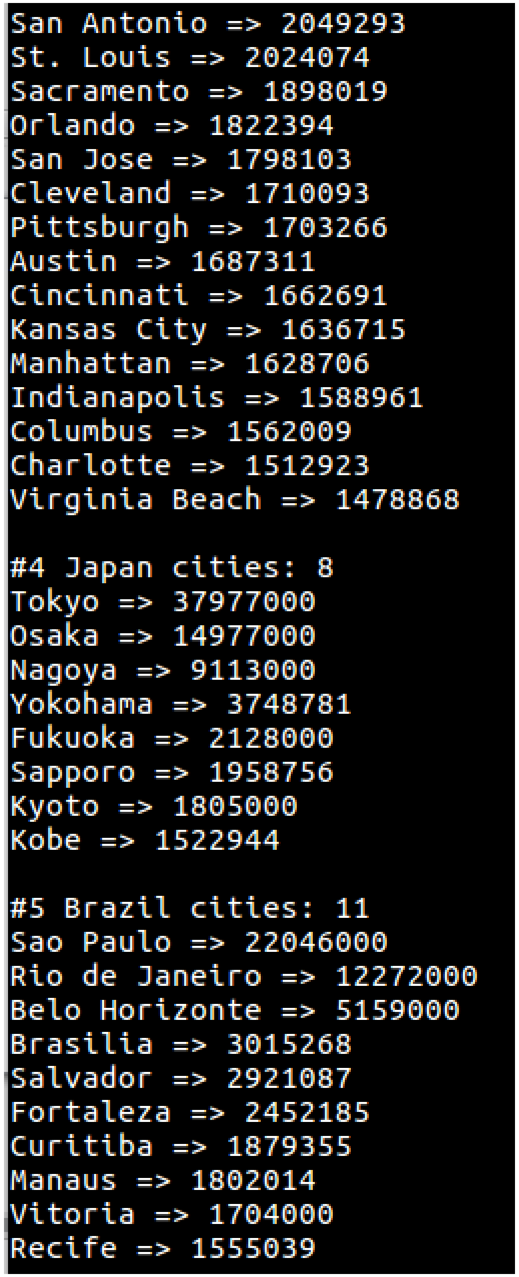
Then, for finding the 5 pairs of cities with the *farthest distances* between each other, we simply need to print the *last 5 elements* of `distanceMap` (why? remember, the map is ordered)

*Note: the function to calculate the distance between cities is provided in the code template.

Previous seminar’s problem - Output Task 1

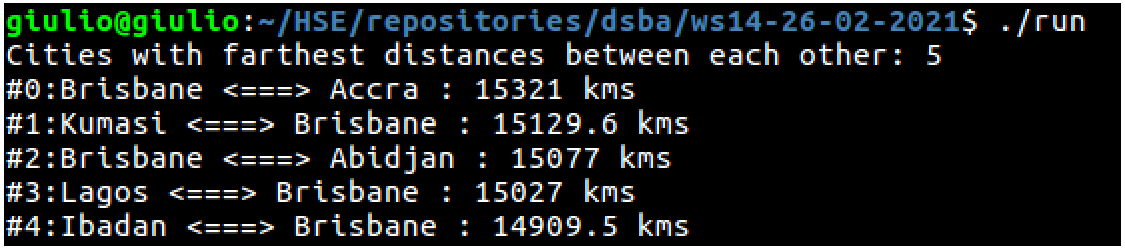


```
giulio@giulio:~/HSE/repositories/dsba/ws14-26-02-2021$ ./run
Countries with most populated cities: 5
#1 China cities: 250
Shanghai => 22120000
Guangzhou => 20902000
Beijing => 19433000
Shenzhen => 15929000
Nanyang => 12010000
Chengdu => 11309000
Linyi => 10820000
Tianjin => 10800000
Shijiazhuang => 10784600
Baoding => 10700000
Zhoukou => 9901000
Weifang => 9373000
Wuhan => 8962000
Heze => 8750000
Ganzhou => 8677600
Tongshan => 8669000
Handan => 8499000
Fuyang => 8360000
Jining => 8023000
Dongguan => 7981000
Chongqing => 7739000
Changchun => 7674439
Zhumadian => 7640000
Ningbo => 7639000
Nanjing => 7496000
Hefei => 7457027
Nantong => 7282835
Yancheng => 7260240
Foshan => 7194311
Nanning => 7153300
Hengyang => 7148344
Xi'an => 7135000
Shenyang => 7105000
Tangshan => 7100000
```



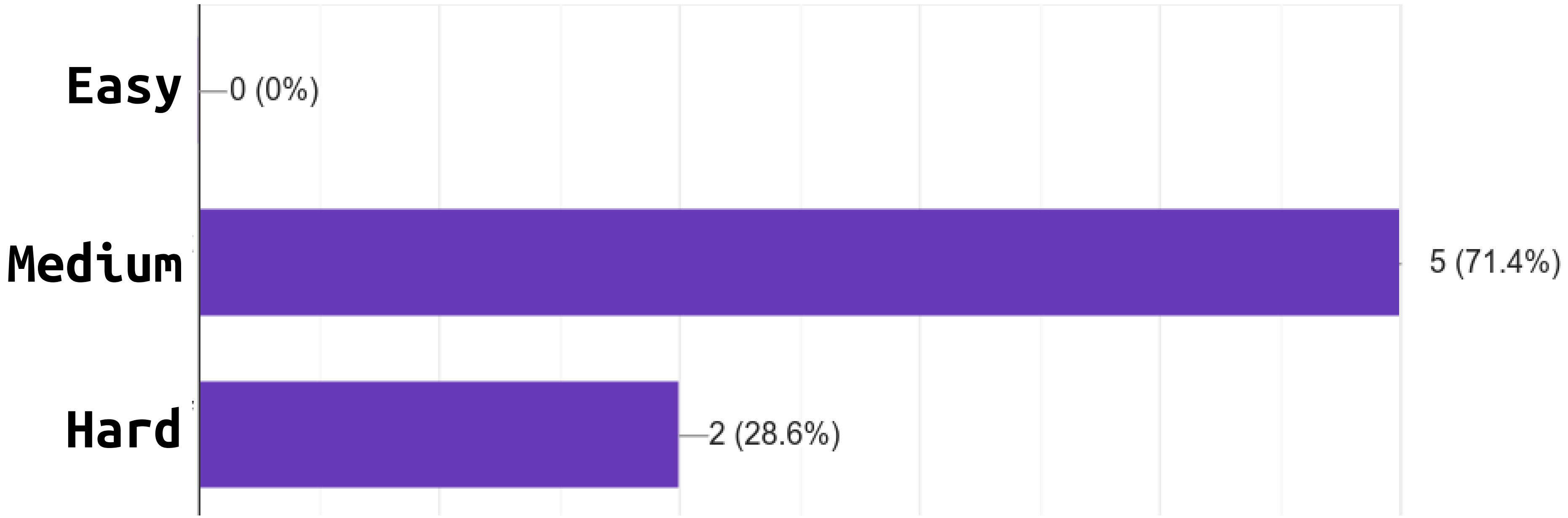
```
San Antonio => 2049293
St. Louis => 2024074
Sacramento => 1898019
Orlando => 1822394
San Jose => 1798103
Cleveland => 1710093
Pittsburgh => 1703266
Austin => 1687311
Cincinnati => 1662691
Kansas City => 1636715
Manhattan => 1628706
Indianapolis => 1588961
Columbus => 1562009
Charlotte => 1512923
Virginia Beach => 1478868

#4 Japan cities: 8
Tokyo => 37977000
Osaka => 14977000
Nagoya => 9113000
Yokohama => 3748781
Fukuoka => 2128000
Sapporo => 1958756
Kyoto => 1805000
Kobe => 1522944

#5 Brazil cities: 11
Sao Paulo => 22046000
Rio de Janeiro => 12272000
Belo Horizonte => 5159000
Brasilia => 3015268
Salvador => 2921087
Fortaleza => 2452185
Curitiba => 1879355
Manaus => 1802014
Vitoria => 1704000
Recife => 1555039
```

Previous seminar’s problem - Output Task 2



```
giulio@giulio:~/HSE/repositories/dsba/ws14-26-02-2021$ ./run
Cities with farthest distances between each other: 5
#0:Brisbane <===> Accra : 15321 kms
#1:Kumasi <===> Brisbane : 15129.6 kms
#2:Brisbane <===> Abidjan : 15077 kms
#3:Lagos <===> Brisbane : 15027 kms
#4:Ibadan <===> Brisbane : 14909.5 kms
```

Students’ Feedback
(7 out of 30 students)



Students’ Feedback
(7 out of 30 students)

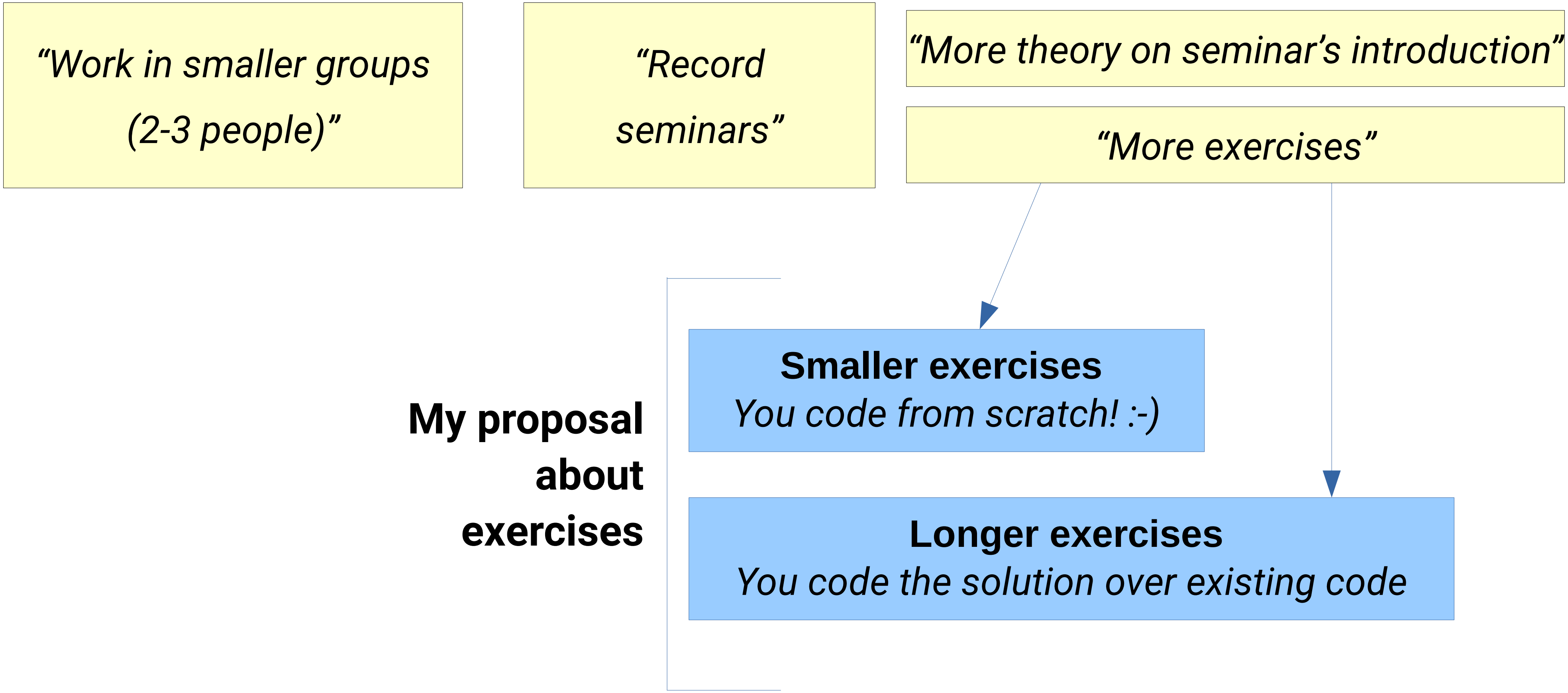
“Work in smaller groups (2-3 people)”

“Record seminars”

“More theory on seminar’s introduction”

“More exercises”

Students’ Feedback



“Work in smaller groups (2-3 people)”

“Record seminars”

“More theory on seminar’s introduction”

“More exercises”

My proposal about exercises

Smaller exercises
You code from scratch! :-)

Longer exercises
You code the solution over existing code

Quick tasks to solve in groups (1 task per group)

Task 1

Create a `std::vector<int> v` of 100 random numbers from the range [1, 10].

- Delete all numbers in the vector which are greater than 5.
- Later, add all the elements of v to a `std::multiset<int> s`
- Print all the elements of the multiset `s`, and also the size of the multiset: `s.size()`.

Task 2

Create a `std::vector<std::pair<int,int>> v of` 100 pairs. For each pair `(xi,yi)` in the vector, `xi` and `yi` are random numbers from the range [1, 10].

- Calculate and print the *epicenter* `ep` of v using the formula:

$$ep = \left(\frac{\sum_{i=0}^{99} x_i}{100} , \frac{\sum_{i=0}^{99} y_i}{100} \right)$$

Task 3

Given the vector

`std::vector<std::string> colors = {“red”, “yellow”, “blue”, “black”, “white”, “green”, “pink”};`

- Create a `std::multimap<std::string,std::string>` of 100 key-value elements , where both keys and values are randomly selected from the vector `colors`.
- Find and print the number of occurrences in the multimap of `(“black”, “white”)` and `(“red”, “blue”)`

Task 4

Fill two matrices `m1` and `m2` of dimension 100x100 random numbers in the interval [1,10].

- Calculate a matrix `m3` as follows: `m3[i][j] = m1[i][j] + m2[i][j] where 0 <= i,j < 100`
- Print `m3`
- Note: you may define a matrix as follows `std::vector<std::vector<int>>`

Structures in C++