Marmara University – Faculty of Engineering – Department of Computer Engineering

Spring 2022 – CSE1242 Computer Programming II
Homework #4

Due: 03.06.2022.Fri 23.29

In this homework, you will write a program that suggests a route to get from a starting point to a target point on a metro line. The details of the program are given below.

A) Please write the following definitions:

1) Define a constant **SIZE** with the value 10.

2) Write a **MetroStation** struct that has to contain a char array of size **20**, named as **name** (example: Haydarpasa); a double **x** and a double **y** that represent the location of a metro station. **typedef** it to **MetroStation**.

3) Write a **MetroLine** struct that has to contain a char array of size **20**, named as **color** (example: red) and a **MetroStation[]** array of size **SIZE**, named as **MetroStations** that contains all metro stations in this metro line ordering from start to end. **typedef** it to **MetroLine**.

4) Write a **MetroSystem** struct that has to contain a char array of size **20**, named as **name** (example: Istanbul) and a **MetroLine[]** array of size **SIZE**, named **MetroLines** that contains all metro lines in this metro system. **typedef** it to **MetroSystem**.

B) Please implement the following functions:

1) Write a function **equals(MetroStation s1, MetroStation s2)** which returns a non-zero value if the name property of the **MetroStation s1** is equal to the name property of **MetroStation s2**; zero otherwise.

2) Write a function addStation which takes two inputs, a **MetroLine*** and a **MetroStation**; and adds the given metro station to the end of the **MetroStations[]** array pointed by the **MetroLine*** pointer. It should return void.

3) Write a function **hasStation** which takes two inputs, a **MetroLine** and a **MetroStation** and returns a non-zero value if the given metro line has a metro station with the same name as the given metro station; zero otherwise.

4) Write a function **getFirstStop** which takes input as a **MetroLine** and returns the **MetroStation** representing the first stop of the given metro line. If there is no such station, your function should return an empty **MetroStation**.

5) Write a function **getPreviousStop** which takes two inputs, a **MetroLine** and a **MetroStation** and returns the previous **MetroStation** to the **MetroStation**

passed as input. If the given station is the first stop on the **MetroLine**, then this function should return an empty **MetroStation**. You may assume that there are no "loops" in the **MetroLine** that is, no station is present on the same line twice.

6) Write a function **getNextStop** which takes two inputs, a **MetroLine** and a **MetroStation** and returns the **MetroStation** after the **MetroStation** passed as input. If the given station is the last stop on the **MetroLine**, then this function should return an empty **MetroStation**.

7) Write a function **addLine** which takes two inputs, a **MetroSystem*** and a **MetroLine** and adds the given metro line to the end of the **MetroLines[]** array pointed by **MetroSystem** pointer. The function should return void.

8) Write a function **printLine** which takes input as a **MetroLine** and prints the metro stations of the given metro line. The function should return void.

9) Write a function **printPath** which takes input as a **MetroStations[]** array and prints the metro stations in the given array. The function should return void.

10) Write a function **getDistanceTravelled** which takes input as a **MetroStation[]** array as path, and returns a double value representing the total distance travelled along a path that goes from the first **MetroStation** in the array, to the second **MetroStation** and so on, until the end of the array. For each **MetroStation** along the journey, it should calculate the distance between the **MetroStation** and the previous one. If the array path contains less than 2 values, your function should return 0.0. You may assume that the variable path is not null. For example, if the array path contains five **MetroStation** your code should calculate the sum of the distance between the first and second stations, the second and third stations, the third and fourth, and lastly the fourth and final stations. (Hint: The distance between two metro stations should be calculated using **x** and **y** coordinates of the metro stations. You should know how to calculate the distance between two points in 2D space.)

11) Write a function **findNearestStation** which takes three inputs as a **MetroSystem**, double **x** and a double **y**, and returns the **MetroStation** which is nearest to the **x** and **y**. To do this, it should look through all the **MetroStations** of all the **MetroLines** inside of the given **MetroSystem** and find the **MetroStation** that has the smallest distance away. You may assume that there is at least one **MetroLine** defined in the **MetroSystem** and every **MetroLine** has at least one **MetroStation** in it. (Hint: You can use the **getFirstStop()** and **getNextStop()** functions to access every entry of a **MetroLine**.)

12) Write a function **getNeighboringStations** which takes three inputs as a **MetroSystem**, a **MetroStation** and a **MetroStation[]** array named as **neigboringStations**, and fills the given **neigboringStations** array containing all neighboring stations to the given station (possibly many if the station is on many lines). For example, if a station is the 3rd stop on the blue line, the 6th stop on the red line, and the 1st stop on the green line, then the function should update the **neigboringStations** containing the **MetroStation** that is the 2nd stop on the blue line, the 4th stop on the blue line, the 5th stop on the red line, the 7th stop on the red line, and the 2nd stop on the green line. Remember that a **MetroStation** will not necessarily be on every **MetroLine**. However, you may assume that the **MetroStation** is on at least one **MetroLine**.

13) Finally, write 2 functions that will help you to find a path from one **`MetroStation`** to another **`MetroStation`** on the given **`MetroStation[]`** array. The first function will take input as two **`MetroStation`** arguments and a **`MetroStation[]`** array you want. The second function will be a recursive function that will take input as a 4^{th} argument.

a) First, write a function **`findPath`** that takes as input a **`MetroStation`** start, a **`MetroStation`** finish and a **`MetroStation[]`** array **`path`**. This function should simply call the **`recursiveFindPath`** function by passing to it as input **`start`**, **`finish`**, given **`MetroStation`** array **`path[]`** and a new, empty **`MetroStation`** array **`partialPath[]`**. The content of the **`partialPath`** will be filled by the **`recursiveFindPath`** function.

b) Write a function **`recursiveFindPath`** that takes input as a **`MetroStation`** start, a **`MetroStation`** finish, a **`MetroStation`** array **`partialPath[]`** and a **`MetroStation`** array **`bestPath[]`**. The content of the **`bestPath`** should contain a full path that goes from start until finish. If no such path exists without requiring "doubling back" (*i.e.* go from A to B and then back to A), then the function should return immediately. The function should return void. To do this, your function should do the following:

i) If **`start`** is contained in the **`partialPath[]`** passed in as input, then your function should return immediately.

ii) If **`start`** and **`finish`** are the same based on the **`equals()`** function then your function should return immediately after setting **`partialPath[]`** as **`bestPath[]`**.

iii) If neither of the above are true, then you should do the following:

(1) Compute a **`MetroStation`** array **`neighbors[]`** of possible places you can get to from **`start`** (You can of course use the function **`getNeighboringStations`** defined before).

(2) For each **`MetroStation`** station in neighbors do the following:

(i) Create a duplicate copy of the array **`partialPath[]`**. Call this copy **`duplicatePath[]`**.

(ii) Add the station **`start`** to the end of the array **`duplicatePath[]`**.

(iii) Calculate the path from the current neighboring station until **`finish`** by using recursion to call the function **`recursiveFindPath`**, but this time with input of *the current neighboring station*, **`finish`**, the **`duplicatePath[]`**, and the **`currentPath[]`**. In the recursive call, the function should update the content of the **`currentPath[]`** given as a parameter to the function (4^{th} parameter named **`bestPath[]`**).

(iv) If the content of the **`currentPath[]`** is not null, then calculate the total distance traveled on this path using the function **`getDistanceTravelled`** that you have written above.

(v) Your function should construct whichever **`bestPath[]`** with the smallest associated distance travelled.

C) Test & Output

1) A sample main function is provided to test your code with this file.

2) The output of your code with the given main function should be as the following:

Sample Output:

```
Metroline red:    Haydarpasa, Sogutlucesme, Goztepe, Kozyatagi, Bostanci, Icmeler.
Metroline blue:   Sogutlucesme, Goztepe, Kozyatagi, Kartal, Samandira.
Metroline green:  Sogutlucesme, Goztepe, Bostanci, Kartal, Icmeler.

The best path from Haydarpasa to Samandira is:
        1. Haydarpasa
        2. Sogutlucesme
        3. Goztepe
        4. Bostanci
        5. Kartal
        6. Samandira
```

3) The nearest metro station to the current location is Haydarpasa (since **`myX=1`** and **`myY=2`**).

4) The nearest metro station to the target location is Samandira (since **`goalX=62`** and **`goalY=45`**).

5) Both Haydarpasa and Samandira stations are not located in a single metro line. The best path (with minimum distance traveled) is given as:

1. Haydarpasa (red line),
2. Sogutlucesme (red line),
3. Goztepe (red line),
4. Bostancı (green line),
5. Kartal (green line),
6. Samandira (blue line).

You do not need to write down the metro lines in the output.

6) If the values for the current and target locations are set as **`myX=9`**, **`myY=4`**, **`goalX=48`**, **`goalY=22`**. Your program should present the following path:

```
The best path from Sogutlucesme to Bostanci is:
        1. Sogutlucesme
        2. Goztepe
        3. Bostanci
```

a) It should be noted that there is a path from Sogutlucesme to Bostanci on the red line; however, it is not the path with minimum distance traveled. Therefore, the suggested path is located on the green line with minimum distance traveled.

7) It should be noted that the content of the main function may change; therefore, test your code with different inputs.

8) You can add more functions than stated above.

9) **It should be noted that <u>selected parts</u> will be graded in your homework.**

SUBMISSION INSTRUCTIONS:

Please zip and submit your files using filename YourNumber_HW4.zip (ex: 150120123_HW4.zip) to http://ues.marmara.edu.tr.

Your program must include necessary comments with your own words to explain your actions!

NOTES:

1) Write a comment at the beginning of each program to explain the purpose of the program.
2) Write your name and student ID as a comment.
3) Include necessary comments to explain your actions.
4) Select meaningful names for your variables and class names.
5) You are allowed to use the materials that you have learned in lectures & labs.
6) Do not use things that you have not learned in the course.
7) **Program submissions** should be done through http://ues.marmara.edu.tr. Do not send program submissions through e-mail. E-mail attachments will not be accepted as valid submissions.
8) You are responsible for making sure you are turning in the right file, and that it is not corrupted in anyway. We will not allow resubmissions if you turn in the wrong file, even if you can prove that you have not modified the file after the deadline.
9) In case of any form of **copying and cheating** on solutions, all parts will get **ZERO** points. You should submit your own work. In case of any forms of cheating or copying, both giver and receiver are equally culpable and suffer equal penalties. **All types of plagiarism will result in zero points from the homework.**
10) No late submission will be accepted.