ASSIGNMENT 8

GENE-121-2017-3F-ASSN-08-Description

Fall 2017

CONTENTS

INTRODUCTION

This assignment is to be done in pairs.

The problems in this assignment are an exercise to:

- solve a problem using 2D arrays
- create a small class in order to control the way a variable is accessed and manipulated

QUESTION 1 – ROBOTIC GEOCACHING (C++)

Prof. Hulls has built a flying geocaching robot and gone to the Canadian Rockies with it. She needs your assistance planning the most energy efficient route to collect all the geocaches. All the geocaches are in an area measuring 2400m (west to east) by 4900m (north to south).

She has two files with relevant data. The first file (elevations.txt) contains elevations in meters. These elevation measurements were taken in a grid pattern, with measurements 100 meters apart. The second file (caches.txt) contains boolean (0/1) values indicating if there is a geocache at a particular location. The location grid of the second file matches that of the first file.

The following diagram illustrates the first few elevations from the most north-west corner of the elevations.txt file.

```
129 <--100m-->  27 <--100m--> 159 <--100m-->  11
 ↑              ↑              ↑              ↑
100m           100m           100m           100m
 ↓              ↓              ↓              ↓
306 <--100m--> 358 <--100m--> 214 <--100m--> 278
 ↑              ↑              ↑              ↑
100m           100m           100m           100m
 ↓              ↓              ↓              ↓
119 <--100m--> 348 <--100m--> 386 <--100m--> 389
```

The data for the cache location from the caches.txt file (for the same corner) looks like the following:

```
0 0 0 0
0 0 1 0
0 0 0 0
```

This means that there is a geocache located where the elevation is 214m. There are no more than 25 geocache locations.

The robot will start at the north-west corner of the grid (where the elevation is 129m). From there, it should move to the geocache location to travel to, that would take the least amount of energy. For each geocache location, it should then look for the next location that would take the least amount of energy. It should never return to a geocache location where it has already been.

The robot uses seven times more energy to travel vertically than it does travelling horizontally. Since the robot flies, it expends the same amount of energy moving up (increasing altitude) as it does moving down (decreasing altitude). You should assume that the robot flies in a straight line between cache locations. In other words, you do not have to worry about mountains getting in the way of the flight path.

Output the list of coordinates showing the path of travel of the robot. For example,

```
(0, 0)
(14, 12)
(8, 4)
```

Note that the above are not actual solutions to the given data. It is simply to show the format of the output.

PART (A) FUNCTION TO READ IN ELEVATIONS FILE

This function receives an already opened elevations file and passes back an array with the data from the file.

PART(B) FUNCTION TO READ IN GEOCACHE LOCATION FILE

This function receives an already opened geocache location file and passes back an array with the locations of the caches. If helpful, this function can return the number of cache locations read from the file.

hint: You may wish to take extra time to consider at least two different ways of storing the geocache information, then choose the one which would likely make the searching easier.

PART (C) FUNCTION TO CALCULATE ENERGY REQUIRED

This function receives the elevations data, a starting coordinate, and an ending coordinate. It will return the amount of energy for the robot to fly from the starting coordinate to the ending coordinate.

Note that we do not need to specify a particular unit of measurement for the energy. The units all cancel out when comparisons are made during the search function.

PART (D) FUNCTION TO FIND NEXT GEOCACHE

This function receives elevations array, geocache location array, and current location. It passes back the location of the geocache location that takes the least amount of energy to travel to. It also returns a boolean value indicating whether there is another geocache location or not. If there is another geocache location, the function returns true. Otherwise, it returns false and it does not modify the geocache location parameters.

PART (E) FUNCTION TO GENERATE OPTIMAL PATH

This function receives the elevations array, geocache location array. It passes back an array containing a list of the coordinates that the robot should travel. This function will also need to either receive the total number of caches as a parameter, or calculate the total number of caches and return that number.

PART (F) FUNCTION TO WRITE PATH TO FILE

This function receives an already opened output file, an array containing a list of coordinates that the robot should travel, and the total number of caches. It will output the list of coordinates to the file.

PART (G) MAIN FUNCTION

The main function should open the necessary files, establish the core data elements (arrays and variables), call the necessary functions, and close all files.

You are welcome to create any additional helper functions that you find useful. Submit your code and output file.

QUESTION 2 – FUEL GAUGE CLASS

It is recommended that you read this entire problem description before beginning to code.

Write a class FuelGauge that tracks and manages a jet aircraft fuel gauge. Include any private member variable(s) that are necessary to the operation of the class. This class should not have any public member variables.

INITIAL DISCUSSION

After reading through the problem description, discuss the following questions with your partner.

- What private member variables are needed?
- What are the units of measurement that the system uses to track the amount of fuel?
- What is the default capacity of the fuel tank?
- What should the smallest allowable fuel tank size be?

Record (write down) your answers to these questions and submit them with your code.

DEFAULT CONSTRUCTOR

The default constructor should initialize the private member variable(s) to appropriate value(s).

DATA CONSTRUCTOR

The data constructor should initialize the private member variable(s) to the given value(s). If the given value(s) is/are invalid, the private member variable(s) should be set to -1.

ACCESSOR – GET FUEL LEVEL

Create an accessor that accepts as a parameter the units of measurement. The accessor returns the amount of fuel in the given units of measurement. If the unit of measurement that are passed to the function is invalid, the function should return a value of -1. Acceptable units of measurement are:

- "L" – liters
- "kg" – kilograms
- "lb" – pounds

You may need to look up a reference for the density of jet fuel.

MUTATOR – REFUELING

Write a mutator method that receives a quantity of fuel as well as a unit of measurement.

This method returns false if the parameters are invalid and returns true if the fuel system is able to add the desired amount of fuel. Note that if the parameters are invalid, the state of the fuel level should not change.

MUTATOR – ENGINE FUEL CONSUMPTION

Write a mutator method that receives a quantity of fuel as well as a unit of measurement.

This method returns false if the parameters are invalid and returns true if the fuel system is able to draw down the desired amount of fuel. Note that if the parameters are invalid, the state of the fuel level should not change.

PRIVATE FUNCTION – UNIT CONVERSION

Write a private function that receives a unit of measurement along with a (floating point) quantity. The function returns the quantity converted into the unit of measurement that the system uses to track the fuel level.

If the received unit of measurement is invalid, the function returns a value of -1.

Note that you should find it useful to call this function from the accessor, mutators, and data constructor.

MAIN FUNCTION

Write a main function to test your class. Include all three test cases given below, in the same main function. Submit your output with your code.

TEST CASE 1

Instantiate your class with the default constructor.

Add a valid amount of fuel to the system.

Remove a valid amount of fuel from the system.

Output the fuel level in all three valid units of measurement.

TEST CASE 2

Instantiate your class with the data constructor using valid parameters.

Remove an amount of fuel from the system that is more than the system currently holds.

Add an amount of fuel to the system that would result in the fuel level being greater than its capacity.

Output the fuel level.

TEST CASE 3

Demonstrate one more way that your class protects the data, that has not already been demonstrated by the previous two tests.