Where in the world is CS?

Etude 9

COSC326

**Problem Summary**

This task for this etude was to create a program that can read and a name (if given) and coordinates from stdin and then convert these coordinates into standard form. Then the program should output these coordinates with their given names on a map using GeoJSON.

**Approach**

**Overall plan**

To approach this problem I first analysed what the etude was actually asking us to do. From this I realised the problem had three main parts: splitting the input into coordinate 1, coordinate 2 and name, converting the coordinates to standard form and writing to a GeoJSON file so that the coordinates could be represented on a map.

**Initial implementation**

Initially I implemented a program that took input one set of coordinates at a time. While I was very happy with this implementation I realised after further reading that this was not the implementation required for the etude. Therefore, I included this implementation as a feature and added the proper implementation which allows you to input coordinates line by line or copy and paste a set of coordinates into stdin. It would also be possible to read from a file, I would just have to add one more else if statement to my inputType method that is the same as the All implementation except it reads from a file one line at a time.

**Invalid coordinates handling**

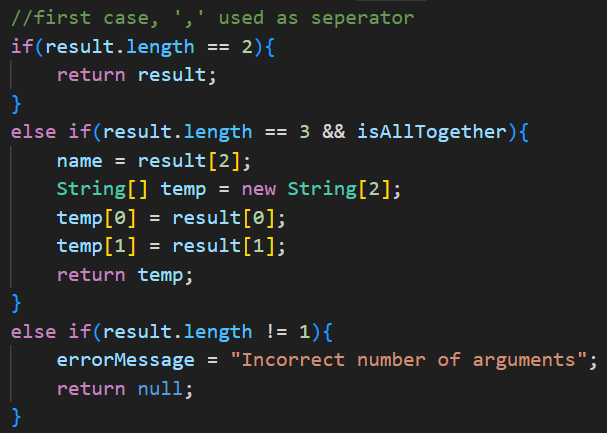
I decided from the start that the easiest way to check the validity of the input was to do it as I went. Every time any unexpected input occurred, I would set a variable errorMessage to a description of the error. The program would check if errorMessage was not null at the end of one of my methods and if it wasn’t it would output the error message and ask the user to input new valid coordinates.

**Step 1: Splitting the data and checking validity of inputs**

For my first implementation I figured it was a better idea to ask for the name first (if they wanted one) and then ask for the coordinates as it would make it easier to split them. I worked on the assumption the only two seperators a user would use is space and , and from there I created three allowed inputs. First was input separated by commas, if there were too many or too few commas an error would occur. Second was split by space, this only worked though if there were no spaces inbetween the individual coordinates (e.g., spaces between degrees and minutes). Therefore, for my third allowed separator was spaces even with spaces between individual coordinates, however this only worked if the coordinate included NSEW declarations. And then finally I allowed for spaces separating both coordinates and degrees/minuets/seconds with no NSEW and with proper DMS characters as in this input I could check the location of ‘ to separate. The other inputs could result in error or the program combining two numbers.

For my second implementation I just edited this split method and added the implementations that worked when the user inputted coordinate, coordinate name. It was a lot more restricted as the name parameter can have spaces in them and NSEW characters. Therefore, I only allowed separated by commas, separated by spaces (with no other spaces meaning name had to be one word in this case), multiple spaces with NSEW coordinates if they were at the end of the coordinates. This option allowed for error if there was an NSEW coordinate in the name, but this would be identified in later stages. And finally, it could split if there were correctly formatted DMS coordinates. However, I asked the user to user commas as this was the least restrictive option and allowed any coordinate and name input assuming the coordinates were valid.

First case



Second case:

A screen shot of a computer code

Description automatically generated with low confidence

Third case:

A picture containing text, screenshot, software, display

Description automatically generated

Fourth case:

A screen shot of a computer code

Description automatically generated with low confidence

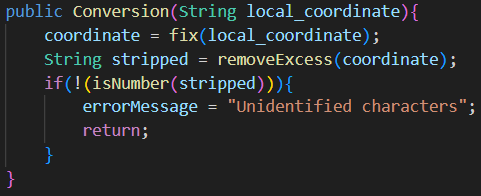
**Step 2: Checking validity of coordinates**

To check the coordinates validity I called the method is valid:

A screen shot of a computer program

Description automatically generated with low confidence

This method creates two instances of the class Conversion, one for each coordinate. This class focuses on checking the validity of the coordinates and returning the coordinates in standard form. In the method above I first create two instances, set the initial latitude for the first to true and the second to false (as this is the required order, it’ll change if there are NSEW present). I then check to see if there are no current errors from the construction of coordinates shown below:



This constructer sets the global variable coordinate to a fixed local\_coordiante. The method fix simply turns some characters that could’ve been inputted into characters that I use in this class, for example, it turns “d” to “°”, I can add code to this at any time if I find any other unexpected characters that are supposed to represent other expected characters. I then create a string from removeExcess(coordinate). This method removes every expected character meaning a valid coordinate would only have numbers left, therefore, when I check if the stripped variable is a number it can only be true for coordinates with valid characters (thus, why I called the fix method earlier.

The isValid method then compares types, if they are the same continue, else return error. After this check it calculates the standard coordinate value for each coordinate checking that the latitude and longitude are in the correct spot and checking they aren’t the same.

Finally, the method does final error checks (in case there were error with the calculations of the coordinates to standard form), and then checks to make sure each coordinate is in its valid range.

**Step 3: Create a GeoJSON file from these coordinates and names**

It took me a long time to figure out how to take this information and put it into a GeoJSON file as I couldn’t find any ways to convert the data using some extension. Instead, in the end, I just decided to create strings in the correct format for features and feature collections and print this to a file instead. The methods for feature creation, feature collection creation and file creation were as follows:

Feature creation:



Feature collection creation:

A picture containing text, screenshot, font

Description automatically generated

File creation:

A screen shot of a computer code

Description automatically generated with low confidence

This part of the etude was relatively simple once I figured out that this was a possible way to do it. However, the hardest part was probably getting the indenting right as this took a lot longer than anticipated.

**Step 4: Printing to GeoJSON.io**

The final step took a lot longer than I thought. Originally I assumed once I created my file I could just write this file to a url with geojson.io and it would output my coordinates. However, geojson.io only accepts a string with the entire feature collection in it. The string also had to be encoded which meant all symbols had to be replaced with their hexadecimal form. The two methods for this process were as follows:

A screen shot of a computer program

Description automatically generated with low confidence

Thankfully the geojson.io website explained exactly what input it accepted, so from there I just had to create a way to make that input from my file which was what getEncodedURL did. In retrospect I could’ve skipped the whole writing to a file step as it was not needed, however, I like the feature that you can see your geoJSON file after running the code.