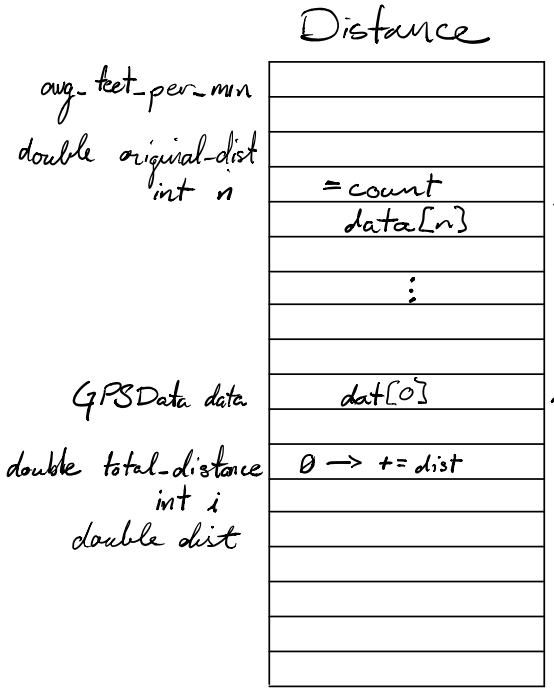
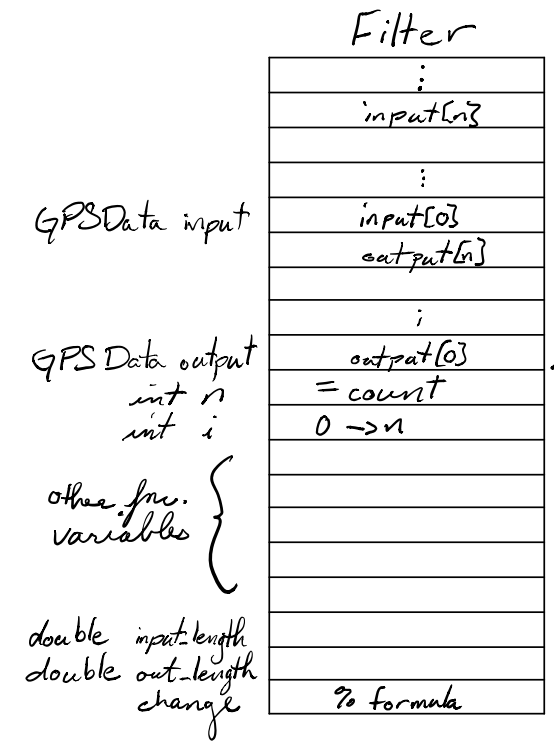


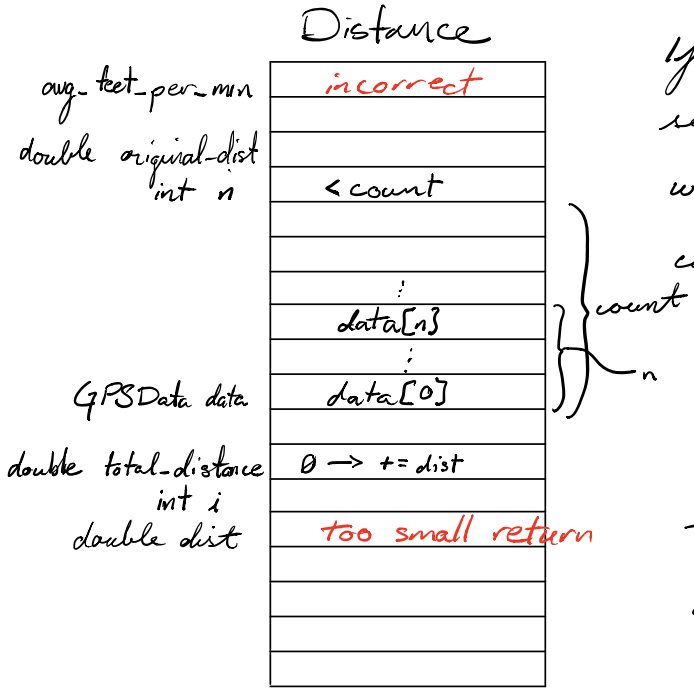
This is how the Distance function should work. The data array is the size n, and the distance output will be correct. Every GPSData object in data will get a value for speed.



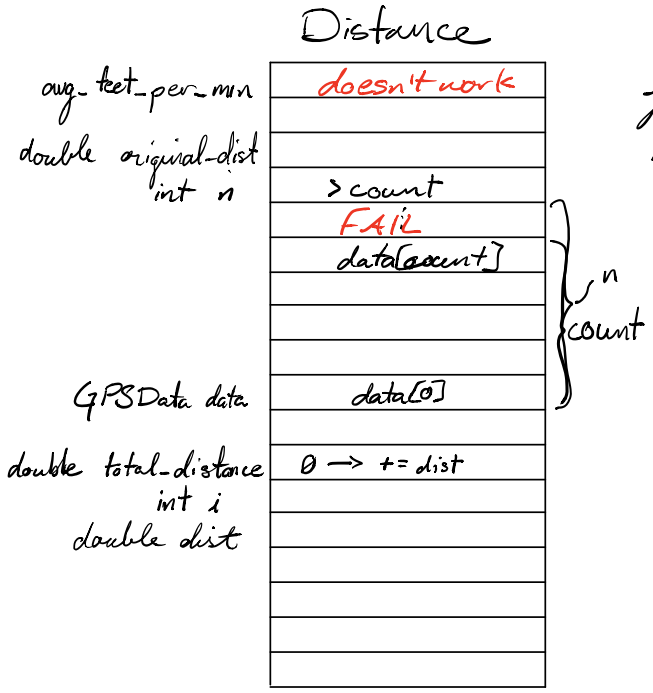
The filter function as it's meant to be. Both input and output are the same size as the original data array.

The input array is read and coordinates averaged correctly, then put in the output array.

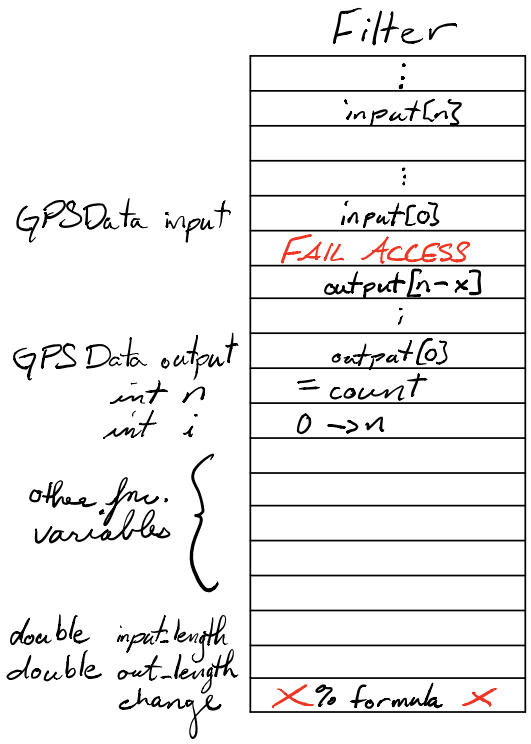
The new path lengths are read and compared.



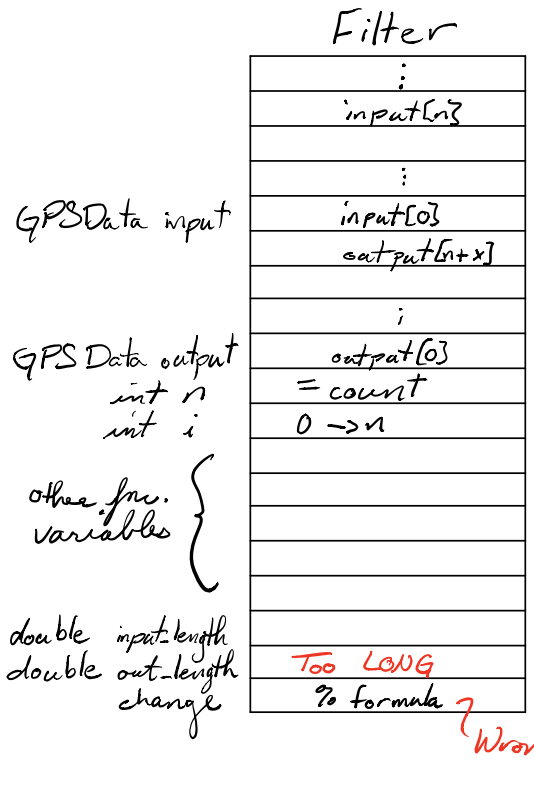
If $n <$ count, n is less than the size of the data array. This will cause the total distance calculated to be too small, and will not write speed values to all of the GPSData objects in data. The returned average speed will be incorrect.



If $n >$ count, then the function will attempt to access values beyond the end of the array, which in the best case does nothing but return a bad average speed. More likely, there will be a memory error and the program will not be able to run as it attempts to access bad values.



In the case where the input array to filter is larger than output, things get messy. The final data object of output will be set to that of input's last, and as the function loops through input and averages position, the loop will eventually try to add GPSData beyond the scope of output, resulting in error as a GPSData is added to memory not allocated for it (or it's attempted).



In the case where output > input in size, the first GPS objects will be the same, but the final one in output will show a longer travel time in the end, and longer distance total, leading to an incorrect % change in path length. Whatever data in the output array that can't be written to will have default values, but this will still mean incorrect distance and average speed values.