**Title:** *Histrogram*

**Name(s):** *Chanasorn Howattanakulphong*

Student number(s) : 65011277

email address(es): [65011277@kmitl.ac.th](mailto:65011277@kmitl.ac.th)

**Aim:**

*The aim is to practice creating functions for calculating values that can be reused is the future when needed. All this is to see how accurate are our GPS from the data provided.*

**Methods:**

*Since this lab is kind of a continuation from the previous GPS lab, I started the work by copying the struct and functions from the old file so I wouldn’t have to write the same identical file again. I then write the code for the input part, which is just reading the provided csv file and storing them into vectors. After that I work on the functions on finding min, max, mean, standard deviation, standard errors and printing each one of them. Lastly, is to create the histogram.*

**Results:**

Text

Description automatically generated

**Discussion:**

*The results are as I expected them to be since everything is already provided for us. (Both the data and the formulas) We just need to create functions based on them.*

**Conclusion:**

*The program works well and satisfy our needs, which are the functioning functions that calculates the details we got to see how accurate the location provided from our GPS is.*

**Acknowledgments:**

*I acknowledge the help and guidance from my teachers, friends, and Tas.*

**Appendix:**

use std::fmt;

use std::fmt::Formatter;

use std::io;

use std::io::\*;

use std::\*;

fn main() {

    let mut text = fs::read\_to\_string(r"D:\Main\Work\KMITL\Rust\lab\13\src\GPSA.csv")

    .expect("Should have been able to read the file");

    let mut gps\_vec: Vec<GPS> = Vec::new();

    text = text.replace(" ", "");

    for x in text.lines(){

      let n = x.split(",");

      let mut str\_vec = n.collect::<Vec<&str>>();

      gps\_vec.push(GPS {

        lati: str\_vec[0].parse::<f64>().unwrap(),

        long: str\_vec[1].parse::<f64>().unwrap()

      });

    }

    let (meanLat, meanLong) = meanGPS(&gps\_vec);

    let (minLat, minLong) = minGPS(&gps\_vec);

    let (maxLat, maxLong) = maxGPS(&gps\_vec);

    let (sdLat, sdLong) = sdGPS(&gps\_vec);

    let (seLat, seLong) = seGPS(&gps\_vec);

    println!("Mean Latitude: {:.5}", meanLat);

    println!("Mean Longitude: {:.5}", meanLong);

    println!("Min Latitude: {:.5}", minLat);

    println!("Max Latitude: {:.5}", maxLat);

    println!("Min Longitude: {:.5}", minLong);

    println!("Max Longitude: {:.5}", maxLong);

    println!("Standard Deviation Latitude: {:.5}", sdLat);

    println!("Standard Deviation Longitude: {:.5}", sdLong);

    println!("Standard Error Latitude: {:.5}", seLat);

    println!("Standard Error Longitude: {:.5}", seLong);

    printHis(&gps\_vec);

}

#[derive(Debug, Clone)]

struct GPS {

  lati: f64,

  long: f64

}

fn his( *lat\_long*: &Vec<GPS> ) -> (Vec<[f64; 2]>, Vec<[f64; 2]>) {

  let dec = 0.00001;

  let mut lati\_bins: Vec<[f64; 2]> = Vec::new();

  let mut long\_bins: Vec<[f64; 2]> = Vec::new();

  for x in *lat\_long* {

    let mut exist = false;

    for y in 0..lati\_bins.len() {

      if (x.lati - lati\_bins[y][0]).abs() < dec {

        lati\_bins[y][1] += 1.0;

        exist = true;

      }

    }

    if exist == false {

      lati\_bins.push([x.lati, 1.0]);

    }

    exist = false;

    for y in 0..long\_bins.len() {

      if (x.long - long\_bins[y][0]).abs() < dec {

        long\_bins[y][1] += 1.0;

        exist = true;

      }

    }

    if exist == false {

      long\_bins.push([x.long, 1.0]);

    }

  }

  (lati\_bins, long\_bins)

}

fn printHis( *lat\_long*: &Vec<GPS> ) {

  let (lati\_bins, long\_bins) = his(*lat\_long*);

  println!("Latitude Histogram");

  for x in lati\_bins {

    let star = "\*".repeat(x[1] as usize);

    println!("{}: {}", x[0], star);

  }

  println!("Longitude Histogram");

  for x in long\_bins {

    let star = "\*".repeat(x[1] as usize);

    println!("{}: {}", x[0], star);

  }

}

fn meanGPS(*gps* : &Vec<GPS>) -> (f64, f64) {

  let mut meanLat = 0.0;

  let mut meanLong = 0.0;

  let mut count = 0.0;

  for x in *gps*{

    meanLat += x.lati;

    meanLong += x.long;

    count += 1.0;

  }

  return (meanLat / count, meanLong / count);

}

fn minGPS(*gps* : &Vec<GPS>) -> (f64, f64) {

  let mut minLat = *gps*[0].lati;

  let mut minLong = *gps*[0].long;

  for x in *gps*{

    if x.lati < minLat {

      minLat = x.lati;

    }

    if x.long < minLong {

      minLong = x.long;

    }

  }

  return (minLat, minLong);

}

fn maxGPS(*gps* : &Vec<GPS>) -> (f64, f64) {

  let mut maxLat = *gps*[0].lati;

  let mut maxLong = *gps*[0].long;

  for x in *gps*{

    if x.lati > maxLat {

      maxLat = x.lati;

    }

    if x.long > maxLong {

      maxLong = x.long;

    }

  }

  return (maxLat, maxLong);

}

fn sdGPS(*gps* : &Vec<GPS>) -> (f64, f64) {

  let meanLat = meanGPS(*gps*).0;

  let meanLong = meanGPS(*gps*).1;

  let mut sdLat = 0.0;

  let mut sdLong = 0.0;

  let mut count = 0.0;

  for x in *gps*{

    sdLat += (x.lati - meanLat).powf(2.0);

    sdLong += (x.long - meanLong).powf(2.0);

    count += 1.0;

  }

  return ((sdLat/count).sqrt(), (sdLong/count).sqrt());

}

fn seGPS(*gps* : &Vec<GPS>) -> (f64, f64) {

  let sdLat = sdGPS(*gps*).0;

  let sdLong = sdGPS(*gps*).1;

*//let count = gps.len() as f64;*

*//return (sdLat/count.sqrt(), sdLong/count.sqrt());*

  return (sdLat \* 111139.0, sdLong \* 107963.0);

}