Lab 2

Project 0
If you have a crufty old code from your advisor, improve part of it.

Project 1

Download the file vabirds.csv.

1. Create a derived type bird_data in a module bird_dat. This is still a type; we do not need a class for this exercise. It should be in its own module, however.

Attributes:
One member of the type will be the species (a character variable) and the other member will be the list of observations. This should take the form of an allocatable array.

Methods:
A. Write a subroutine to act as the constructor for the type. Its parameters will be the species name and the array of input data. Allocate the type array based on the size of the input array.

B. Write a stats method that takes only an instance of the type and returns the mean and standard deviation of the observations for that instance.

C. Write a minmax method that takes an instance of the type and the array of years and returns the maximum observed, the minimum observed, and the years for maximum and minimum. You may use the maxval, minval, maxloc, and minloc intrinsics.

2. Write a main program that uses your module and also uses the sorters module that you can download (sorters.f90). This implements bubblesort. Bubblesort is simple and slow but is more than sufficient for this exercise. Note that the subprogram is destructive, i.e. it overwrites the array to be sorted, so make a copy if you don't want that.

Remember to write an explicit interface for each subprogram in this "main" file. Do not use `contains`.

Read the file name from the command line. First of all you will need to count the number of lines in the file. Write a function count_lines that does this and returns the number. It is up to you whether you pass it the number of header/footer lines or whether you will apply that information later. Your read_data routine will then need an interface for the count_lines function. Count_lines can check for the existence of the file and return 0 if it is not found.

Still in read_data, using the number of items in the file, corrected for the header and the two footers, allocate an array of bird_data types. Loop through this array calling your constructor for each species. The read_data routine should return the array of years and the array of bird_data types.

Request a species name from the user. Find the species in your array of types and print its mean, standard deviation, and results from minmax.

Compute an array of the means for all species. Use the pbsort routine from sorters to sort this array. This procedure also returns the *permutation vector*, which is an array of the indices of the original positions. For example, if after the sort the permutation vector is (17,3,55,11,23, and so forth) that means that the element that was previously 17 is now the first in the new array, and so on. Note that these sorters return in ascending order (smallest to largest).

From the sorted mean array and the permutation index, print the names of the 10 most common (by mean) species over the years of observations. Hint: you can use a trick to reverse a dimension of an array in Fortran: R=A(ndim:1:-1)

Test the user input portion for

TurkeyVulture
TuftedTitmouse
ElegantTrogon

Project 2

Download the file bodyfat.csv. This is a dataset of body fat, age, height, and weight for a set of participants in a study.

BMI categories are as follows:

Severely underweight	BMI < 16.0
Underweight	16 <= BMI < 18.5
Normal	18.5 <= BMI < 25
Overweight	25 <= BMI < 30
Obese Class I	30 <= BMI < 35
Obese Class II	35 <= BMI <40
Obese Class III	BMI > 40

Write a bmistats module containing functions/subroutines for the following:

1. Convert pounds to kilograms. Use the actual conversion factor, not the approximate one. Look it up on Google.
2. Convert feet/inches to meters. Look up the conversion factor, do not guess at it.
3. Compute BMI
4. Determine where the user falls in the table supplied and return that information in an appropriate form.

Write a module stats that implements the following:
1. mean of an array
2. standard deviation of an array
3. outlier rejection using Chauvenet's criterion. Pseudocode given further down.
Make as much use of Fortran intrinsics/array operations as you can.

Write a main program that implements the following:

1. uses your modules
2. reads the input file into appropriate allocatable arrays (use one-dimensional arrays for this project). Don't assume you know the length of the file (but you can assume the number of header lines is fixed).
3. pass appropriate arrays to a subroutine that computes an array of BMI data based on height and weight and returns the array.
4. Rejects the outlier(s). The function should return an array of logicals that you can apply to the original data using where (Fortran) or similar. Create new arrays with the outlier(s) deleted.
5. Print a crude histogram of BMI values. To do this, divide the data into an appropriate number of bins as defined by the categories, obtain the count for each bin, and print out a line of asterisks representing the number of BMIs in each bin. It should look something like the example below (I have not computed the results, this just sketches the general appearance of the output)
6. Write a file that contains the corrected data for age, weight, height, bodyfat, and BMI. Use Excel or whatever to plot BMI as a function of percentage body fat. Be sure to plot it as a *scatter plot* (points only, no connecting lines).
7. Create a namelist file and use it to read in the cutoffs for the various bodyfat categories. Pass that information to the routine that computes the BMI categories.

Histogram example:

```
*****
*******
*************
***********
**********************
```

****
****

Chauvenet's criterion:

It's not the state of the art but works pretty well.

1. Compute the mean and standard deviations of the observations.
2. Compute the absolute values of the deviations, i.e. abs(A-mean(A))/std(A)
3. Use the tails devs=devs/sqrt(2.)
4. Compute the probabilities prob=erfc(devs)
   Fortran: erfc is an intrinsic in any fairly recent compiler.

5. The criterion is that we retain data with prob>=1./(2*N_obs) (number of observations)

Project 3

Convert your type from Project 1 into a class.