Lab 10 - Data

Due: end of class

OVERVIEW

For this lab, you will do some basic manipulations of data. For this, you will need to do some File I/O and use structs (along with other data structures already covered in other labs).

TASKS

The goal here is to keep track of population information about the countries of the world.

Task 0:

First, we need to decide the appropriate data structure for storing the data. Your instructor has supplied you with a file (populations.csv), which is a simplified version of data sourced from the UN (a few countries with non-ASCII characters in their name were left out just in case your program has problems with those). Save the file to $PWD.

The format of the file is as follows:

- Columns 1-5: populations (in 1000s) of country in 1950, 1970, 1990, 2010, 2015, respectively. Since the population is in 1000s, the number is a real number.
- Rest of line: Name of country.

We wish to keep track of the populations in those years, as well as 1960, 1980, and 2000.

Thus, a suitable data structure might be:

```
struct Country {
  string name;
  double pops[8]  // stores population in 1950, 1960,..., 2010, 2015 in order
};
struct World {
  int     numCountries;
  Country countries[MAXCOUNTRIES];
} myWorld;
```

You will change this data structure later.

Task 0:

Write a function to initialize myWorld to data read from the supplied data file. There are a few complications:

- Reading the populations is straightforward, and can be done using the standard file IO functions we covered in class (i.e., ">>" using an input stream). However, since the names of some countries contain spaces, we need to use getline instead of >> for the name field. Fortunately, the country is the final field so we can use ">>" to read the populations and then a getline to finish the line.
- You will need to input data until the end of file is reached. Recall that getline's return value is false if at end of file, so its easy to check for this.
- For years in which no statistics are available (e.g., 1960), you should store an estimated population computed to be the average of the previous and next decade.

To check that you did this correctly, you may wish to use ddd to check the value of myWorld.

Task 1:

Write a program that computes the growth rates of all countries between 1960 and 2015. Also modify the above data structures as follows:

- Store the growth rates so that they don't need to be recomputed every time you need them.
- Store the indices of the fastest and slowest growing countries (over 1960-2015).

Obviously, you need to decide which struct is more appropriate for these new fields.

Your program should then output:

```
The fastest growing country is XXXXX, which grew by XXXXX% between 1960 and 2015.
The slowest growing country is XXXXX, which shrunk by XXXXX% between 1960 and
2015.
```

Task 2:

Now we wish to print the names of all countries that had negative growth rates over that period, ordered by 2015 population. We will do this in 2 steps:

1. Define another data structure to store information about just these countries. Of course, the only data structures you know so far are arrays and structs, so you should use one/both of them.
2. Since this data structure should be sorted (by 2015 population), scan through World and insert the appropriate countries into this new data structure. You probably want to think about why an insertion sort is appropriate here.

Finally, print all the countries in this new data structure as follows:

```
The countries that shrank between 1960 and 2015 (and their 'growth' rates) are:
  country1 XXXXX%
  country2 XXXXX%
  ...
```

HAND IN

Your 136 instructor will tell you what to hand in and how.

GENERAL COMMENTS FOR ALL PROGRAMS THIS SEMESTER

You should have the following header on all programs:

```
/*
  Author: <name>
  Course: {135,136}
  Instructor: <name>
  Assignment: <title, e.g., "Lab 1">

  This program does ...
*/
```

GRADING

All 135 and 136 programs this semester will be graded on:

- Correctness: Does your program work?
- Testing: Have you generated sufficient and good test data to give reasonable confidence that your program works?
- Structure: Have you structured your code to follow proper software engineering guidelines? This includes readability and maintainability.
- Documentation: How well documented is your code? Good documentation does not repeat the code in English, but explains the point of each code block, highlighting any design decisions and/or tricky implementation details.