Homework 3: Data Wrangling

The United States Social Security Administration releases data dating back to 1880 on babies' first names, genders, and their popularity.[1] Similarly, the Centers for Disease Control releases data on life expectancy in the United States.[2]. In this assignment, you'll be putting these two data sources together to answer vital questions such as "What is the most popular baby name in 2007?", "Approximately how many Emmas are alive today?", and so on.

To do so, you'll use Scala's built-in data structures (lists, sets, and maps) to write several data-processing functions that can be composed together to answer complicated queries. Although you can write all these functions directly using recursion, the point of the assignment is to sequence Scala's built-in methods together to write them quickly and compactly. Most of the functions only need 1 line of code, if you exploit the right library features.

In addition, you'll be working with CSV files (*comma-separated values*), which is the format in which a lot of public data sets are released.

Preliminaries

You should create a directory-tree that looks like this:

```
./wrangling
├── build.sbt
├── project
│   └── plugins.sbt
└── src
    ├── main
    │   └── scala. ..... Your solution goes here
    └── test
        └── scala ..... Yours tests go here
```

Your `build.sbt` file must have exactly these lines:

```
libraryDependencies += "edu.umass.cs" %% "csv" % "1.0"
```

The `project/plugins.sbt` file must have exactly this line:

```
addSbtPlugin("edu.umass.cs" % "compsci220" % "1.0.0")
```

You'll also need to download two datasets:

- The CDC data on life expectancy:

  https://www.cs.umass.edu/~arjun/courses/compsci220/cdc-life-expectancy.csv

- The SSA data on baby names:

  https://www.cs.umass.edu/~arjun/courses/compsci220/ssa-births.csv

You should place these files in the `wrangling` directory that you created above.

The Data

In a CSV file, each line has a row of data, where each row has a sequence of columns, separated by a commas. For example, the first few rows of `ssa-births.csv` are:

[1] https://www.ssa.gov/oact/babynames/limits.html
[2] http://www.cdc.gov/nchs/products/life_tables.htm

```
1880,Mary,F,7065
1880,Anna,F,2604
1880,Emma,F,2003
1880,Elizabeth,F,1939
```

We can interpret this data as "In the year 1880, 7065 female babies were born named Mary; in the year 1880, 2604 female babies were born named Anna" and so on. In fact, the file has over 1.8 million lines of entries like this, with data as recent as 2014. *If this file is too large for your computer to handle, you should delete some entries.*

The file `cdc-life-expectancy.csv` is much shorter and has entries that look like this:

```
2010,76,81
2009,75,80
2008,75,80
```

We can interpret this data as "The average life expectancy of U.S. babies born in 2010 is 76 years for males and 81 years for females", and so on.[3]

The `build.sbt` file for this assignment is configured to load a library for reading CSV files. For example, the program:

```
import edu.umass.cs.CSV
CSV.fromFile("cdc-life-expectancy.csv")
```

Produces the value:

```
List(
  List("2010", "76", "81"),
  List("2009", "75", "80"),
  List("2008", "75", "80"),
  ...)
```

Programming Task

Your programming task is to implement the functions described below. You should place all functions within an object called `Wrangling`.

1. Write a function that consumes a list of SSA records and only produces those records from a given year:

   ```
   def yearIs(data: List[List[String]], n: Int): List[List[String]]
   ```

2. Write a function that consumes a list of SSA records and only produces those records for years greater than the given bound:

   ```
   def yearGT(data: List[List[String]], bound: Int): List[List[String]]
   ```

3. Write a function that consumes a list of SSA records and only produces those records for years lower than the given bound:

   ```
   def yearLT(data: List[List[String]], bound: Int): List[List[String]]
   ```

4. Write a function that consumes a list of SSA records and only produces records with the given name:

   ```
   def onlyName(data: List[List[String]], name: String): List[List[String]]
   ```

5. Write a function to calculate the most popular name in the given dataset and the *total* number of children born with that name.

   Hint: It is likely that children are born with the same name in several years. So, you should first calculate the total number of childen with each name.

   ```
   def mostPopular(data: List[List[String]]): (String, Int)
   ```

6. Write a function to calculate the number of children born in the given dataset.

[3] The CDC releases more precise statistics. The numbers have been cleaned up to make the assignment a little easier.

```
def count(data: List[List[String]]): Int = ???
```

7. Write a function that produces a tuple with the number of girls and boys respectively.

   ```
   def countGirlsAndBoys(data: List[List[String]]): (Int, Int)
   ```

8. Writen a function to calculate the set of names that are given to both girls and boys.

   ```
   def genderNeutralNames(data: List[List[String]]): Set[String]
   ```

9. Write a function to determine if a person with the specified `gender` and born in the specified year (`birthYear`) is expected to be alive (in the year `currentYear`), according to the CDC life-expectancy data.

   If `currentYear` is the last year the person is estimated to be alive, be optimistic and produce `true`.

   The CDC data only ranges from 1930—2010. Therefore, assume that `birthYear` is in this range too.

   ```
   def expectedAlive(gender: String, birthYear: Int, currentYear: Int): Boolean
   ```

10. Write a function to estimate how many people from the given dataset will be alive in a particular year:

    ```
    def estimatePopulation(data: List[List[String]], year: Int): Int
    ```

You'll notice that many functions consume and produce a list of births. The key idea is that you can compose these functions together to pose complex queries. For example, suppose `allBirths` holds the complete list of births:

```
val allBirths = CSV.fromFile("ssa-births.csv")
```

- The following query calculates how many girls and boys were born between 1990 and 2000:

  ```
  countGirlsAndBoys(yearGT(yearLT(allBirths, 2001), 1989))
  ```

- The following query tells is if there were more Emma's born in 2010 than in 2009:

  ```
  count(onlyName(yearIs(2010, allBirths), "Emma")) > count(onlyName(yearIs(2009, allBirths), "Emma"))
  ```

- The following query reports the most popular name of 2013:

  ```
  mostPopular(yearIs(2013, allBirths))
  ```

- The following query estimates the number of John's alive today who were born after 1980:

  ```
  estimatePopulation(yearGT(allBirths, 1980), 2015)
  ```

Hand In

From the `sbt` console, run the command `submit`. The command will create a file called `submission.tar.gz` in your assignment directory. Submit this file using Moodle.

For example, if the command runs successfully, you will see output similar to this:

```
Created submission.tar.gz. Upload this file to Moodle.
[success] Total time: 0 s, completed Jan 17, 2016 12:55:55 PM
```

Note: The command will not allow you to submit code that does not compile. If your code doesn't compile, you will receive no credit for the assignment.