**Line-Based File I/O: Searching For A Name**

**The goal for this exercise** is to make sure that you can open up a file of line-based data records, search through the records for a particular name, and then display some information about that record.

For this exercise you will process a file with data obtained from the Social Security Administration. They provide a web site showing the distribution of names chosen for children over the last 100 years in the US (<http://www.ssa.gov/OACT/babynames/>).

Every 10 years, the data gives the 1000 most popular boy and girl names for kids born in the US. The data can be boiled down to a single text file as shown below. On each line we have the name, followed by the rank of that name in 1900, 1910, 1920, ..., 2000 (11 numbers). A rank of 1 was the most popular name that year, while a rank of 1000 was the least popular. A 0 means the name did not appear in the top 1000 that year at all. The lines are in alphabetical order; you may use this fact to make your implementation of this exercise more efficient than you could if the file was unsorted.

...

Sam 58 69 99 131 168 236 278 380 467 408 466

Samantha 0 0 0 0 0 0 272 107 26 5 7

Samara 0 0 0 0 0 0 0 0 0 0 886

Samir 0 0 0 0 0 0 0 0 920 0 798

Sammie 537 545 351 325 333 396 565 772 930 0 0

Sammy 0 887 544 299 202 262 321 395 575 639 755

Samson 0 0 0 0 0 0 0 0 0 0 915

Samuel 31 41 46 60 61 71 83 61 52 35 28

Sandi 0 0 0 0 704 864 621 695 0 0 0

Sandra 0 942 606 50 6 12 11 39 94 168 257

...

We see that “Sam” was #58 in 1900 and is slowly moving down. “Samantha” popped on the scene in 1960 and is moving up strong to #7. “Samir” barely appears in 1980, but by 2000 is up to #798. The database is for children born in the US, so ethnic trends show up when immigrants have kids.

Your program is to give an introduction and then prompt the user for a name to display. Then it will read through the data file searching for that name. If it finds it, it should list the rank for that name in each of 1900, 1910 … 2000. If the name is not found, then the program should generate a short message indicating that. The following example transcripts (with user input in **bold**) demonstrate the format that should be followed:

|  |
| --- |
| Example Of Finding A Name: |
| Name to search for?**Sam**Found Sam! Year Rank 1900 58 1910 69 1920 99 1930 131 1940 168 1950 236 1960 278 1970 380 1980 467 1990 408 2000 466 |
| Example Of **NOT** Finding A Name: |
| Name to search for?**Zog**Zog was not found |

**What you need to do for this exercise:**

1. Implement the program as described above
	1. The function named Search inside the SSA class is pretty much implemented for you. The only exception is that it uses the other, overloaded version of Search to implement the above functionality.
	2. The overloaded version of Search (also inside the SSA class) is given two parameters: a string containing the name to search for, and a string containing the name of the file to search within. This function must print out the output described in the tables above AND it must return an integer array. If the name is NOT found in the file then this function can simply return null. If the name IS found, then the array must contain all the ranking data found in that file for that name, where slot 0 contains the data for 1900, slot 1 contains the data for 1910, etc. Continuing the example from the table above, asking for the name “sam” would cause this Search function to both print the above output and to return an array that contains:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (Index) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| (Rank) | 58 | 69 | 99 | 131 | 168 | 236 | 278 | 380 | 467 | 408 | 466 |

(This array will be used by the test functions to verify that you’re getting the right results)
(Note that you may assume that the array will always be 11 elements in length)

* 1. You should compare the names in a case-insensitive manner. So you should produce the “Example of finding a name” output for Sam (as shown above), sam, sAM, SaM, etc, etc
	2. Your functions must be able to read the files located in the “Files\Exercise\_Files” subfolder. While you’re obviously free to rearrange the files on your personal computer while you’re developing the various parts of this exercise you are required to be able to read the file named “Files\Exercise\_Files \SSA\_Names\_Long.txt” when you hand in the exercise
	3. As an aside, it is safe to use the return statement to get out of a ‘using’ statement in C#: <http://aspadvice.com/blogs/name/archive/2008/05/22/Return-Within-a-C_2300_-Using-Statement.aspx>

*Thanks to Stuart Reges (of UW-Seattle), and to Nick Parlante of Stanford for the original (and full) versions of this assignment*

*Also see:*[*http://www.nytimes.com/2003/07/06/magazine/06BABY.html*](http://www.nytimes.com/2003/07/06/magazine/06BABY.html)