CPE 101
Fall 2009
Project 4

Due Date

- **Friday, November 13th by 11:59pm– remember, no late projects accepted!**

Objectives

- To practice writing loops in C
- To practice using arrays in C.
- To practice using structures in C.
- To practice using arrays of structures in C.
- To practice writing functions in C.
- To practice using strings in C.
- To practice writing programs which use more complex data representations and the control structures to do computations (such as searching) on those structures.

Ground Rules

- Your program must not use any global data.
- Your program must implement the required functions as specified.
- Your program must use the required functions appropriately.
- Your program must mimic the sample program's behavior ***exactly*** and in all cases.

Orientation

Against all bureaucratic stereotypes, the Social Security Administration provides a neat web site showing the distribution of names chosen for kids 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 997 was not very popular. A 0 means the name did not appear in the top 1000 that year at all. The elements on each line are separated from each other by a single space. The lines are in alphabetical order, although we will not depend on that.

```
...
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.

You will be creating a program that allows the user to print out information about the popularity of particular baby names over the last hundred years. Your program will read in data from a file about the popularity of close to 4500 baby names. Your program will then allow the user to search for specific baby names and then graph the popularity of that name. For example the graph for Samantha would look like this:

```
Name: Samantha
Popularity by year:
1910    1920    1930    1940    1950    1960    1970    1980    1990    2000
----------------------------------------------------------------------------
-       -       -       -       -       ^       ^       *       *       *
----------------------------------------------------------------------------
Key: * = very popular, ^ = common, + = less common, - = rare
```

Your program will also allow the user to select to print out the entire list of data. See the sample output files for details.

Note this program was inspired by the “NameSurfer” assignment from Stanford’s nifty csc assignments (http://nifty.stanford.edu/)

Section 0: Develop Incrementally…

There are many reasonable ways to approach developing this program. Immediately beginning to write code is not a recommended approach! The first and most important thing to do is to understand what the program is required to do. To do so, run the sample program, read the entire specification, discuss the project with your instructor, as necessary, until you feel you understand what is required of the program. You may want to write some pseudo-code for parts of this program before you begin to write actual C code. Once you have done this much, the writing of the code should be almost trivial (well, eventually it will be!).

When you do begin writing code, you should decide on some incremental steps you can code and test. There are many reasonable ways to approach this problem incrementally. One suggestion is to create a record to represent the data and make sure you are comfortable printing the data in your record correctly, by making a function to print out a name record. Next it would be good to focus on reading the name data from the file into your record data structure. Test that you have read in the data correctly using the print function you’ve developed

Section 1: Required Function Specifications…

You are required to develop the following functions. Think about what values these functions will need to consume (their input) and their return type (output). Please name them reasonable names.

Implement a function which prints out the popularity graph for a given baby name. This function will allow for two different types of printing. One is just straight numeric values (see the below example). And one is using ascii characters to represent the names popularity (see the above example). Specifically, in the character graph, '*' is for names in the top 100 names for that year, '^' is for names in the top 500 (but not in the top 100), '+' is for names in the top 900 names (but not in the top 500), and '-' is for names ranking in the final category. Note that we will begin printing the graph for 1910, not 1900.

Implement a function, which prints out the entire contents of the baby data in the format shown in the sample output files. Be sure to use the function you just wrote that prints out a popularity graph for a single name. For example a small portion of the output table would look like this:

```
…
Name: Zona
Popularity by year:
1910    1920    1930    1940    1950    1960    1970    1980    1990    2000
------------------------------------------------------------------------
707     879     0       0       0       0       0       0       0
------------------------------------------------------------------------
Name: Zora
Popularity by year:
1910    1920    1930    1940    1950    1960    1970    1980    1990    2000
------------------------------------------------------------------------
574     849     908     0       0       0       0       0       0
------------------------------------------------------------------------
Name: Zoraida
Popularity by year:
1910    1920    1930    1940    1950    1960    1970    1980    1990    2000
------------------------------------------------------------------------
0       0       893     584     576     689     0       0       0       0
------------------------------------------------------------------------

…
```

Section 3: The main function…

Write a main function, which reads in the data from the data file (called names-data.txt), which is available in the class examples file on the class webpage. Then write a main loop, which allows the user to choose what they would like to do, search for a specific name or print the table. The user can keep searching for different names until they are done. For example, when the program first starts off it says:

```
Welcome to the name popularity program
```

```
Reading in name data file
Done!  Read in 4429 records
If you would like to search for a name, enter 1.
To print the entire list of names, enter 2.
To exit the program, enter -1.
Enter choice:
```

Section 4: Testing against input

I highly recommend testing your program using an input file. You can redirect input from standard input (the console) to instead come from a file. You can do this by using the < on vogon. For example:

```
12:01pm vogon ~>a.out < input_file.txt
```

where input_file.txt is the name of a text file that contains the input for your program. Example input files will be on the class webpage.

In addition, you can re-direct output to a file as well, using > on vogon. For example:

```
12:01pm vogon ~>a.out < input_file.txt > output_file.txt
```

where output_file.txt is the name of a text file that contains the output generated by your program. Example output files will be provided at a later date.

Finally, you can use the **diff** command on vogon to display any differences in two files. For example:

```
12:01pm vogon ~>diff file1.txt file2.txt
```

will print out any differences between file1.txt and file2.txt. Once you have written your program, run it with the sample input files and save the output and use the diff command to make sure all your calculations are as expected.

Section 5: Handing in Your Source Electronically…

1. Move the necessary file(s) to your vogon account using your favorite FTP client program.

2. Log on to vogon using your favorite Shell client program.

3. Change directory (cd-command) to the directory containing the file(s) to hand in.

4. Be sure to compile and test your code on vogon using the required compiler flags (-Wall –ansi – pedantic) one last time just before turning the files in.

5. Use the following handin command

handin zwood csc101p4 babynames.c

6. You should see messages that indicate handin occurred without error. You can (and should) always verify what has been handed in by executing the following command:

```
handin zwood csc101p4
```