EEN118 LAB ELEVEN

In this lab, you will be performing some important data-processing operations, specifically sorting a large database file. Sorting data is a very important operation in computing for many reasons. One of those reasons is that it makes the data more accessible to humans once it is printed (imagine trying to use a telephone directory in which the names do not appear in any particular order). Another reason is that it makes the data more quickly searchable by the computer (recall the Binary Chop algorithm).

There are four large datafiles to download for this lab. You will only need the first one unless you get on to the advanced parts. They are all available on the class web-site, and are named `database1.txt`, `database2.txt`, `database3.txt`, and `database10.txt`.

Download the file "`database1.txt`" and use a text editor to take a quick look at it. You will see that it contains data about a number of people. Each line contains exactly four items: a person's social security number, their first name, their last name, and their date of birth. The four items are separated by spaces, but no item will ever contain a space. Here is a sample from the middle of the file:

```
243810667 Chester Peters 19210320
244260287 Lynne Dobson 19781211
244550439 Napoleon Stein 19810110
244940274 Eileen Holloway 19351104
245340593 June Ride 19370522
246230419 Rupert Ogham 19590810
248890854 Christopher Nixon 19510503
250410626 Lars Root 19520508
252190308 Petunia Aspen 19421001
253780249 Otto Osmond 19270802
257390263 Roscoe Smithers 19840718
258080395 Ellery Farmer 19370524
258230892 Calvin Hornswaggle 19431217
259280426 Tammy Moriarty 19490204
259320410 Jim Wilder 19441217
264880013 Azalea Smelley 19740811
266640093 Incitatus Laurel 19720124
267110552 Isaac Mason 19661121
```

The names are all randomly generated, so there is no confidential data in there. As you may have noticed, the date of birth is provided as a single integer, in the format `yyyymmdd`; Chester Peters was born on the 20th of March 1921.

The file `database1.txt` contains exactly 1000 lines of data.

1. ***(Basic)*** Read the Data

   Write a program that creates appropriate arrays large enough to hold all the data, then reads all the data from the file into those arrays. There should be one array for social security numbers, another array for first names, another array for last names, and a fourth array for birth dates. Make your program close the file, then print out the first 10 items of data from the arrays, so that you can make sure everything was read correctly.

2. ***(Basic)*** Basic Search

   Make your program ask the user to enter a name. It should then search through the data in the arrays (don't read the file again), finding any entry with a matching name. Correct matches with either first *or* last name should be accepted. For every matching entry that is found, print out all four data items: the social security number, first and last names, and date of birth of each matching person.

   Remember that if you use the == operator to compare strings, the test is *case-sensitive*. The user (i.e. you) will have to type the name exactly correctly, with capital letters in the right places.

   Important: Good clean design will make this lab *much* easier. Write a separate function that searches the arrays; do not put all the work in `main`.

3. ***(Basic)*** Find the Oldest

   Modify your program so that after closing the file, instead of printing the first ten items of data, it searches through them all to find the oldest person represented. It should print the social security number, first and last names, and date of birth of the oldest person found.

   Important: As for part two, good clean design will make this lab *much* easier. Write a separate function that searches the arrays to find the oldest person; do not put all the work in `main`.

4. ***(Basic)*** Promote the Oldest

   For some unfathomable reason, the management wants the oldest person to occupy the first position in the arrays. Modify your program so that after finding the oldest person, it swaps his or her data with the data already occupying the first position in the arrays. Remember that the first position in an array is numbered zero, not one.

5. ***(Intermediate)*** Now Promote the Second Oldest.

   The management has now decided not only that the oldest person must occupy the first positions in the array, but also that the second-oldest person must occupy the second position in the array. So, after searching for the oldest and oving their data to the front of the array, now search the *remainder* of the array (all except the first element), and move the oldest person you find (which must be the second oldest of all) into the second position of the array. Make sure you swap data, so that whoever was originally in the second position is not lost.

6. ***(Intermediate)*** More of the Same.

The management are going to keep on adding requirements like this, next putting the third-oldest in the third position, then the fourth, then the fifth. There is no knowing when they will grow out of this petty obsession, so make things easier for yourself. Modify your search function so that it can be told how much of the array to search. That is, give it two int parameters (let's call them `a` and `b`); its job is now to search only the portion of the arrays between position `a` and position `b`, to find the oldest person therein. This makes it very easy to search the remainder of the array to find the second and third oldest.

7. ***(Intermediate)*** The Ultimate Demand.

Now the management make their final demand. You are to repeat the process of moving the n^{th}-oldest person into the n^{th} position 1000 times. (remember, 1000 is the number of data records in the whole file).

This will result in the arrays being completely sorted. Do it, and check that it worked. Make your program print the contents of the four arrays after it has finished. Look at the outout to make sure that everyone is printed in order of their age.

8. ***(Intermediate)*** Sorting the File.

Once you have sorted the contents of the arrays, it might be a good idea to save the sorted data in a file. Make your program create a new file, and write all the contents of the arrays into that file in a sensible format. Use a text editor to look at the file and verify that it has the same format as the original file, and all the data is properly sorted.

9. ***(Advanced)*** How Fast Is It?

It is important to know how long computer operations are going to take when they have to work on a large amount of data. `library.h` contains two functions: `look_at_the_clock`, and `double_time`. The first of them does exactly what its name suggests. The second returns the time that was seen when it last looked at the clock, in seconds, as a `double`. The time it returns is accurate to about a millisecond.

Use this function (twice) to time how long it takes the computer to sort the arrays of 1000 data items. Do not include the time it takes to read the file or the time it takes to write the new file, just the pure sorting time. Note the time that you observe.

Now you know how long it takes to sort a database of 1000 items. How long do you think it would take to sort a database of 2000 names? 3000 names? 10,000 names? Think about those questions, and work out what you believe the answer is. *Then* find out what the real answer is. The three other files, `database2.txt`, `database3.txt`, and `database10.txt` contain 2000, 3000, and 10000 data items respectively. If your program was nicely written, it will be a few seconds' work to change the array sizes and make it read a different file.

See how long it takes to sort these larger files, and compare the results to your predictions. If your predictions weren't substantially correct, make sure you understand why. You have just demonstrated a very important phenomenon of computing.