CSc 127B — Introduction to Computer Science II
Fall 2015 (McCann)

http://www.cs.arizona.edu/classes/cs127b/fall15/

Program #10: Binary Searching a Binary File

Due Date: ***Tuesday*** *December 1st, 2015, at 9:00 p.m. MST*

Overview: Binary Search is normally presented as a technique for searching in–memory data structures. However, it can be effectively used to search for uniformly-sized records in binary files, too.

Thanks to your taxpayer dollars and the requirement that the government conduct a census of the population every 10 years, the U.S Census Bureau collects a large volume of data and makes much of it available to anyone who cares to view it. For this assignment, a bit of it makes great data for an ordered binary file of records. In the source folder on your X: drive you should find a binary file named `county2k.bin` with some info on over 3,200 counties from the 2000 census. Each 78-byte record of that file has data about a specific county (or equivalent unit) in the United States. In this assignment, you'll let the user search the `county2k.bin` file (using a **recursive** binary search) for county names.

Assignment: Write a complete, well–documented Java program named `Prog10.java` that gets from the command line both the path to the `county2k.bin` file and the prefix of a name of a county, and, using a **recursive** binary search, finds (or fails to find) *all* counties within that file whose names begin with that prefix (case doesn't matter; 'A' matches 'a'). By 'prefix,' we mean the initial characters; for example, "P", "pi", "piM", and "PImA" are all prefixes of "Pima County" for this assignment. If the name is found, the program will display to the screen the complete name, state abbreviation and population of *each* county starting with the given name prefix. If the name is not found, the program is to display a message to that effect. Additional output requirements are detailed in the "Output" section, below.

Please be careful to note that the binary search is to be on the `county2k.bin` file of records. You may **NOT** write your program to read all of the records from the file into an array and search the array. You must perform binary search the file itself, reading in just one record at a time.

Data: The file `county2k.bin` can be found on the class web page. If you save it to another computer, make sure that your saved copy is the same size – exactly 251,082 bytes.

Each county and county-like unit is represented in the file by five fields of data totaling 78 bytes. In order, those fields are: A 2-byte state abbreviation (ex: AZ), an `int` value for the state FIPS code (ex: 4), an `int` value for the county FIPS code (ex: 19), an `int` value for the county population (ex: 843746), and a 64-byte string with the name of the county (ex: Pima County). Java's classes for reading from binary files provide methods for reading integers and characters.

You may be wondering how you can conduct a binary search without an array, ArrayList, etc. We know that to conduct a binary search, we need data in sorted order and in a direct-access data structure. The binary file has been created with the county records in order by name, meeting the first condition. Because our county records are each 78 bytes in size, we can compute the first byte of the n-th record just as we computed the first byte of the n-th element of an array: The first record starts at byte 0, the second at byte 78, the third at byte 156, etc. All we need to do is jump to the computed byte, which Java provides methods to do (see the Hints section, below), followed by reads of the fields of the record that starts there.

(Continued ...)

Output: In addition to the list of county names, state abbreviations and populations (for a successful search) or the failure message (for an unsuccessful search), output the following. As the binary search is performed, for each 'probe' into the binary file, display to the screen the values of 'low,' 'high,' and 'mid' (think of these as file record numbers, where the first record in the file is 0), along with the county names of the records associated with each of the three. This will help you (and us!) see how the search progressed. Whether the binary search has succeeded or failed, display the quantity of 'mid' records that were read from the file. This count is not to include the reads of the 'low' and 'high' records, nor the additional reads necessary to find and display any additional counties with the same name.

Here's an output fragment. You don't have to follow this exact format; you just need to display all of the required information in a clear manner:

```
> java Prog10 county2k.bin barb

Probe #1:  Low:     0 (Abbeville County)
            Mid:  1609 (Lake County)
           High:  3218 (Ziebach County)

[...]

The counties whose names are prefixed with the letters 'barb' are:

    KS     5307  Barber County
    AL    29038  Barbour County
    WV    15557  Barbour County

The number of 'mid' records read was [...]
```

Note that "Santa Barbara County" isn't included; its name contains 'barb', but doesn't begin with it.

Turn In: Use 'turnin' to submit your `Prog10.java` file to the `cs127bsXp10` directory at any time before the stated due date and time.

Want to Learn More?

- "FIPS" stands for **F**ederal **I**nformation **P**rocessing **S**tandard.
- `www.census.gov` is the U.S Census Bureau's home page.
- The `county2k.bin` file was created from the Census 2000 Gazetteer files: `https://www.census.gov/geo/maps-data/data/gazetteer2000.html`

Hints, Reminders, and Other Requirements:

- In case you missed it: Your binary search must be ***recursive***.
- A record in a file is just a logical concept describing a physical group of fields. In this assignment, a binary file record has five fields. By reading the five fields in order, you've read a record; easy as that!
- It's been a while, so a reminder is in order: We covered binary file I/O earlier in the semester. Reviewing the example programs, particularly `T03n05.java` and `T03n06.java`, will be helpful.

  You may wish to look at one class in particular that we didn't cover when we talked about binary files: `RandomAccessFile`. It's nice because you can use it to read from as well as write to binary files. Plus, it has a convenient `seek()` method. You don't *need* to use `RandomAccessFile`, but we recommend it.
- We suggest that you start by writing a small program that reads the content of the first record and displays that information to the screen. This will demonstrate that you've figured out how to read a record. Then, extend that program to read all of the records. Next, try reading the records in reverse order (record n, then record $n-1$, etc.). Only when you can do these things should you worry about implementing the **recursive** binary search.
- Big hint: Your program will work if the number of counties in the file changes ... won't it?