Systems Programming Lab #1 - C Programming Basics

1. Goal

- Practice with C programming basics: declaring variables, types, arrays, and functions.
- Writing C code and functions that uses statically declared arrays and structs and arrays of structs.
- Practice writing and using C functions. Pass by value: basic types and array parameters.
- C I/O: scanf, printf, and building a simplified file reading library.
- More practice with top-down design.

2. Lab Description

For this lab, you will implement an employee database program in C. Your program will store employee information in an array of employee structs, sorted by employee ID value. Your program will read in employee data from an input file when it starts-up. Your program should print out a menu of transaction options, read in the user's selection, perform the chosen operation on the employee database, and repeat until told to quit. The options your program must support are the following (**Do not change the numbering of these menu options in your program (i.e. menu option 2 must be look up by ID, option 5 must be quit, etc.)**):

1. Print the Database
2. Lookup employee by ID
3. Lookup employee by last name
4. Add an Employee
5. Quit

Your program should continue handling the user's transaction choices until the user enters 5, the "QUIT" option.

2.1 Program Start-up

Your program will take one command line argument, which is the name of the input file containing employee information. For example, telling your program to load the employee data stored in the file "small.txt" would look like:

```
./workerDB small.txt
```

File Format

The input file format consists of several lines of ASCII text. Each line contains the information for one employee in the following order:

```
six_digit_ID  first_name  last_name  salary
```

For example, here is a file with 4 employees:

```
273836 Edsger Dijkstra 93000
493570 Leslie Lamport 63000
518364 Vint Cerf 85000
998447 Barbara Liskov 75000
```

2.2 File I/O

For this assignment you will develop a small library that includes functions you'll be using later in the assignment to simplify reading data from the input file. Name you library files readfile.c and readfile.h

Here are the functiones that you'll include in the library:

- Open a file by calling: `open_file()`, passing in the name of the file to open as a string; open_file() returns 0 if the file is successfully opened, and -1 if the file cannot be opened.
- Functions to read values of different types into program variables: `read_int()`, `read_string()`, `read_float()`. These functions take arguments much like scanf does: they need to know the memory location of where to put the value read in. For example:

```
•      int x;
•      float f;
•      char s[20];
•      ...
•      ret = read_float(&f);  /* returns 0 on success, -1 on EOF
*/
•      ret = read_int(&x)    /* returns 0 on success, -1 on EOF
*/
•      ret = read_string(s)      /* returns 0 on success, -1 on
EOF */
•
•
```

- Close the file when you are done with it: `close_file()`

2.3 Storing Employee Records

You should define an employee struct with the fields necessary to store information for one employee. Think about which types you want to use for the different fields, and use meaningful field names. Because the DB supports a look-up operation by last name, I suggest storing first and last name values in two separate fields rather than using a single name field. You may assume that no first nor last name is longer than 64 characters, including the trailing '\0' character.

Here is an example of how you might define a person struct to store a single name and age value for each person:

```
  #define MAXNAME  64    /* #define and use constants for values that
don't change */

  struct person {
     char name[MAXNAME];   /* a C-style string (array of chars) */
     int  age;
  };
```

Use the following information about valid field values to help you decide which type to use for individual fields (for some fields there may be more than one reasonable choice):

- The six digit ID value must be between 100000 and 999999 inclusive.
- Salary amounts must be between $30,000 and $150,000 inclusive.
- The salary is a positive whole number amount (no decimals).
- You should assume that every employee has exactly two names (a first and a last), so "John D. Rockefeller" is not a valid name for purposes of this assignment.

The employee data read in from the file should be stored in an array of employee structs. You may assume that there are never more than 1024 employees.

3. Requirements

Your output should look like this sample output (it doesn't have to be an exact match, but it should have a similarly formatted structure). This output was obtained by processing this file. NOTE: employees should be listed in increasing ID order.

For full credit, your submission should meet the following requirements:

- When asked by the user to print the database, you should do so in a tabular format followed by printing out the total number of employees in the database. See the sample output for an example. Your output does not need to be identical to mine, but it should have a similar tabular form.
- When asked by the user to look up an employee by ID, you should use BINARY SEARCH to find a matching employee in the data base. If one is found, print out the Employee information. If not, print out a "not found" message to the user.
- When asked by the user to look up an employee by last name, you may search the database however you like. If no there are no matching employees, print out a "not found" message. If there is more than one employee with a matching last name, it's sufficient to print the information for only one of them (you don't need to find all matches, just one).
- When asked by the user to add an employee to the database, your program should prompt them for the employee's information. It should also ensure that the user enters valid values for each field, and print out an error message and re-prompt the user to try again, repeating until the user enters valid values. It should then print out the field values of the employee to add, and ask the user if s/he really wants to add the employee. If yes, insert the new employee in the array. You must make sure that any employee entered by the user will be assigned an ID that is numerically larger than any existing employee record in the database.
- Your code should be commented, modular, robust, and use meaningful variable and function names. This includes having a top-level comment describing your program and listing your name and the date. In addition, every function should include a brief description of its behavior.

- It should be evident that you applied top-down design when constructing your submission (e.g., there are multiple functions, each with a specific, documented role).

4. Tips

- Before even starting to write code, use top-down design to break your program into manageable functionality.
- Test your code in small increments. It's much easier to localize a bug when you've only changed a few lines.
- Reading in input from the user (e.g., reading menu selections) should be done using `scanf`.
- Use **CTRL-C** to kill a running program stuck in an infinite loop.

5. Submit your work

Here is what you need to deliver on the Blackboard:

- A link to your git repository such that we can clone it, build an executable and test it.

NOTE: please make sure your repository contains a README file that explains how to build an executable and execute it.