Homework 5, TCSS 333 Spring 2016
Due: Saturday, June 4, 9:00 a.m.

OBJECTIVE

The objective of this assignment is to give you more practice with using structures and binary random access files.

ASSIGNMENT SUBMISSION

To get credit for this assignment, you must

- ✓ write a multi-file program in C
- ✓ submit your project as a tar file through Canvas exactly as instructed (naming, compatibility, etc.)
- ✓ submit your assignment on time

PROBLEM STATEMENT

For this assignment, you need to write a program that maintains a database for a local animal rescue as a binary file named `rescue.dat` (provided with the assignment). The program needs to support operations such as (1) adding an animal, (2) deleting an animal, (3) searching for an animal, and finally (4) writing all animals' information to ASCII text file called `rescue.csv`. These operations need to be presented to the user as options (use the numbers given above) and the program can only terminate when the user selects to do so (0). All the operations need to be performed directly on the database file, without storing all animals into a data structure, such as an array. The program can only store one animal at a time, a small short int array of size 10, and a few primitives required by the processing logic.

An animal should be stored and processed using the following structure:

```
struct animal {
    short int id;
    char name[20];
    char species[35];
    char size;
    short int age;
};
```

Since the structure memory layout is different on a 32-bit machine than on a 64-bit machine, you will want to force the compiler to store variables of animal type the same way (using the same alignment / padding) regardless of the platform. In order to do so, use pragma pack directive before your product structure definition, with the value 1

```
#pragma pack(1)
// struct definition
```

Output Files

Your program should produce two output files: an updated `rescue.dat` file, according to the user's operations, and an ASCII output file `rescue.csv` that lists one animal per line and uses commas as delimiters, as in the following example:

```
1,Allegra,Pseudois nayaur,S,5
2,Amela,Cerdocyon thous,M,11
3,Angela,Capra falconeri,L,8
5,Anjolie,Ailurus fulgens,X,10
6,Athena,Moschus fuscus,X,2
8,Ava,Cephalophus jentinki,U,13
```

Note that in this particular example, there are no animals with ids 4 or 7, as one of them passed away, while the other one was released back into its native habitat.

Input/Output File

The binary file, `rescue.dat`, is organized as follows:

- the first entry on the file is a short (16-bit) integer describing the number of available ids
- it is followed by ten short (16-bit) integers that list available id numbers (to be stored in short int array)
- they are followed by animals' information, listed one by one, as shown in the structure defined above; all animals are listed in the increasing order by their id number, starting at value 1; if there is a hole in id numbers, e.g. 4 and 7, then these structures' information is still present in the file, except the name component contains the string "unknown" to signify an empty record.

The binary file based on the example given above, in the text view, would look as follows:

```
2
4  7  0  0  0  0  0  0  0  0
1,Allegra,Pseudois nayaur,S,5
2,Amela,Cerdocyon thous,M,11
3,Angela,Capra falconeri,L,8
4,unknown, Bison bison athabascae,X,15
5,Anjolie,Ailurus fulgens,X,10
6,Athena,Moschus fuscus,X,2
7,unknown, Cerdocyon thous,L,2
8,Ava,Cephalophus jentinki,U,13
```

Processing Logic

Searching for animals

The program is to search for one animal at a time and the search is to be based on the id number entered by the user. Your program should give the user a choice to select an id number and verify it is a valid id. If the id is invalid, or it is an empty record, the program displays an error message. If an animal is found, and it is not an empty record, the program displays this animal's information to the screen. In either case, the program returns to the main menu.

Note that searching needs to use random file access.

Deleting animals

Only one animal can be deleted at a time. Your program should give the user a choice to select an id number and verify it is a valid id. If the id is invalid, or it is an empty record, the program displays an error message and goes back to the main menu. Given a valid id, the program simply changes the animal's name in the file to "unknown" and updates holes information as well. After taking care of deletion, the program goes back to the main menu.

You should keep track of deletion ids, if there is room in the array containing holes numbers, as you should later on use those numbers for the insertion operation.

Note that deletion needs to use random file access.

Adding animals

Only one animal can be added at a time and it is up to the program to select the animal's id. The program is to display the new id number and ask the user to enter name, species, size, and age (in this order, one by one), and store this information in the file in the correct spot. Assume valid user entries, except remember that species could consist of more than one word.

The order of additions algorithm the program is to follow is:

- first fill any existing holes, as indicated by the initial list of values provided in the file (e.g. in our example input file, it will be id values 4 and 7)
- next, fill any existing holes, as collected from the deletion component of the code
- next, since the program is only keeping track of up to 10 holes, fill any additional existing holes

- finally, if there are no holes in the id sequence, the program must append at the end of the database and pick the next consecutive number for the id, e.g. 9, if the database contains 8 animals numbered 1-7

Note that insertion needs to use random file access for all the bullets listed above, other than the 3rd one.

Program Specs

- The minimum number of files for your program is 3 files: .h with structure definition and function prototypes, .c with function definitions, and another .c for a driver
- You need to create a makefile for your code
- Name the executable file `rescuebin`
- Your program has to follow basic stylistic features, such as proper indentation (use whitespaces, not tabs), meaningful variable names, etc.
- You need to comment your code
- You are not allowed to use global variables
- **Your program must compile in gcc gnu 90 – programs that do not compile will receive a grade of 0**
- All program files must reside in one folder and the contents of the entire folder must be compressed using tar utility (check lab 1 directions to make sure you are compressing an entire folder and not individual files)
- Your tar files should be named hw5.tar

Extra Credit (15%)

Open-ended – extend the program in some fashion – explain in comments at the top of your code.

Program Submission

On or before the due date, use the link posted in Canvas next to Assignment 5 to submit your tar file. Make sure you know how to do that before the due date since late assignments will not be accepted.