Programming Assignment 5, TCSS 333A Winter 2021

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 that compiles and you wrote on your own (no copying or help outside of CSS mentors, TLC, or instructor allowed)
- ✓ submit your project as a tar file through Canvas exactly as instructed (naming, compatibility, etc.)
- ✓ submit your assignment by the due date

PROBLEM STATEMENT

For this assignment, you need to write a program that displays a text menu and, depending on the user input, manipulates the binary file called `animals.dat`. Valid input choices are 0-5, where 0 indicates a user wants to quit, while 1-5 indicate they want a particular operation to be performed. After each operation 1-5 is performed, the program is to continue, display the menu again, and react to the new user input.

Menu choices and processing logic

If the user enters 1, the program is to perform processing similar to the *strings* command in Unix (or the string format display in *ghex or xxd*). In other words, the program is to read in an entire binary file in a sequential fashion one byte at a time and for each byte, if the byte content is consistent with an ASCII value 32-126, the program is to write it out to console as an ASCII character. If the value is not consistent with a printable character, the program is to print a period character to the console. Each line in the terminal is to be no longer than 80 characters. After the text version of the binary file is written out to the console, the program is to go back to the initial menu.

If the user enters 2, the program is to perform processing similar to the way numbers are displayed in the *ghex* editor. In other words, the program is to read in an entire binary file in a sequential fashion, four bytes at a time, and for each four-byte unit, it is to write it out to console as a signed integer. When writing to console, make sure you also print a space between all integer values and that the line is no longer than about 80 characters. After the integer version of the binary file is written out to the console, the program is to go back to the initial menu.

For user options 3 - 5 assume that the input file `animals.dat` consists of records of animal type, defined as follows:

```
#pragma pack(1)
struct animal {
    short int id;
    char name[20];
    char species[35];
    char size;
    short int age;
};
```

Since the structure memory layout is different on different machines, the pragma pack directive forces the compiler to store variables of animal type the same way using no alignment.

If the user enters 3, the program is to ask the user which animal record to find and to display to the screen – the search is to be done by the integer denoting the record number. Note that for this search you are to use random file access (i.e. jump directly to that record without sequential file processing) and that the record number and the *id* of an animal are the same entity. For example, if the file consists of entries like:

```
1,Allegra,Pseudois nayaur,S,5
2,unknown,Ailurus fulgens,X,10
3,Athena,Moschus fuscus,X,2
```

then the 3rd record on that file is `3,Athena,Moschus fuscus,X,2`
All animals are listed in the increasing, sequential order by their id number, with no id gaps, starting at value 1. If an animal record is empty, then the structure information is still present in the file, except the *name* component contains the string *unknown* to signify an empty record. If an invalid id is entered, in the example above it would be any value other than 1 or 3, your program is to display an error message. Otherwise, animal record is displayed. In either case, the program is to go back to the initial menu.

If the user enters 4, you to assume that the input file `animals.dat` consists of records of animal type, as described above, and the program is to ask the user which animal record to update and allow the user to update it. The search component should be implemented the same way as in option 3 and the user should be allowed to update any part of the record other than the animal id. If a valid id is used, an update is to be performed by asking the user which piece of information they want to update:

- If letter *n* is entered, name is to be modified
- If letter *p* is entered, species is to be modified
- If letter *s* is entered, size is to be modified
- If letter *a* is entered, age is to be modified

After the user enters the letter to indicate which part of the record they want to modify (only one field can be modified at a time), prompt for the updated value, and write an updated information back to the file. After that go back to the initial menu of 0-5. Bear in mind that animal names and species could consist of more than one word but assume it will be no longer than the char array set-up. An update should make use of random file access and not process an update if an invalid id or letter is entered but rather display an error message and go back to the initial menu.

If the user enters 5, you are to assume that the input file `animals.dat` consists of records of animal type, as described above, and the program is to ask the user which animal record to overwrite – indicated by the record number. Assuming a valid value has been entered, the program is to overwrite the animal entry with the information from the last record in the file (other than the id) and then shrink the actual file so that it does not contain the last record For example, if the file consists of entries like:

```
1,Allegra,Pseudois nayaur,S,5
2,Anjolie,Ailurus fulgens,X,10
3,Athena,Moschus fuscus,X,2
```

then if the user typed in 1, the resulting file would be:

```
1,Athena,Moschus fuscus,X,2
2,Anjolie,Ailurus fulgens,X,10
```

Make sure you use random file access for this menu choice. Look at function *truncate* or *ftruncate* to help you shrink the file. If an invalid id is entered, in this example any value outside of 1-3 range, your program is to display an error message. In either case, the program is to go back to the initial menu.

At the end of the program, when the user enters 0 to quit, write modified `animals.dat` contents to `animals.csv` file skipping all unknown animals (sample provided). Note that `animals.dat` itself, after the program run, should contain contents of the modified original file.

Program Specs

- The binary file is to be open only once per program run, i.e. do not keep reopening the same file for different value choices
- Do not assume that the exact same file will be used for grading your program, i.e. your program needs to be scalable and not dependent on the number of initial animals in the `animals.dat` file that is given to you for program development
- The only data types you are allowed to use in this program are primitive data types, *struct animal* and data types comparable to its *struct animal* components (e.g. small strings). You are NOT allowed to use arrays or any other data structures in your code since with binary files one can treat the file itself as if it were a data structure
- Your program should use several functions (e.g. one function per menu choice)
- You are NOT allowed to use global variables (global constants allowed)

- You do NOT need to create an ADT for an animal but you should split the program into minimum 3 files, in which case .h should contain the structure definition and all function prototypes required by the program, .c should contain all function definitions, and *driver.c* that should contain *main* function only
- You need to create a makefile for your code that is to be called **makefile**
- Name the executable file *animalbin*
- 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
    - Remember to include your name and a statement whether you tested your code on the cssgate server or Ubuntu 16.04 LTS Desktop 32-bit
- **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 called *prog5* and the contents of the entire folder must be compressed using tar utility (check tutorial 1 directions to make sure you are compressing an entire folder and not individual files)
- Your tar files should be named *pr5.tar*

GRADING

Remember, the programming assignments are graded as pass / no pass only and 85 points (out of 100 are needed to pass).
Options 0 (with csv file writing), 1, and 2 are worth 10 points each
Option 3, 4, 5 are worth 15 points each
Proper decomposition into a 3-file program structure is worth 15 points
Proper coding style, meaningful identifiers, etc. is worth 10 points

PROGRAM SUBMISSION

On or before the due date for assignment 5, use the link posted in *Canvas* next to *Programming Assignment 5* to submit your tar file. If you have a valid reason for missing the due date, contact the instructor – it is the end of the quarter and depending on your circumstances, we may or may not accept late submissions (with or without penalty).