Assignment 7: File I/O

CS3423 - Systems Programming — Rocky Slavin - UTSA

For this assignment, you will use **C**'s I/O functions to create a simple tenant database for an apartment complex. The system will store basic information about apartments and their tenants and allow the user to create, read, update, and delete them. All information for all apartments will be stored as **binary records in a *single file***.

This assignment requires only the utilities used so far in the I/O lecture notes. **Do not** use bash, sed, awk, find, grep, Python, or any programming languages or utilities besides the C binary functions used in class. Only binary I/O functions should be used to store data to the filesystem (it is OK to use other functions when prompting the user).

Storing Apartment Information

All apartment information will be stored in a single binary as record of the following structure where **`firstName`** is the first name of the tenant with no spaces, **`lastName`** is the last name of the tenant, which may contain spaces, **`leaseStart`** is the starting lease date, **`leaseEnd`** is the ending lease date, and **`balance`** is the tenant's current balance.

```
1 typedef struct
2 {
3     char firstName[32];
4     char lastName[32];
5     char leaseStart[16];
6     char leaseEnd[16];
7     int balance;
8 } Apartment;
```

The program will store all apartments using the above struct in a single filed called **`apartments.dat`** within the same directory as the program (do not assume this file exists). All apartments will be referenced by a zero-index apartment number. Apartment records will be stored in **`apartments.dat`** in apartment number order. Note that the apartment number will be specified by the user when an apartment is entered and will not necessarily be sequential. If **`apartments.dat`** does not exist, one should be created by the program.

Program Execution

When the program is run, the following should occur.

1. Upon running your program, the user should be presented with the following menu:
   ```
   Enter one of the following actions or press CTRL-D to exit.
   C - create a new apartment record
   R - read an existing apartment record
   ```

```
U - update an existing apartment record
D - delete an existing apartment record
```

2. The user then enters a one-character action (upper or lowercase), leading to one of the following.
   - `C`: an apartment record is created
     (a) From the terminal, read the following one at a time
        i. Apartment number **(zero-indexed integer)**
        ii. First name
        iii. Last name
        iv. Lease start date (string with slashes)
        v. Lease end date (string with slashes)
     (b) Using the values entered by the user, write the binary structure to `apartments.dat` in the appropriate position. **The starting balance should be 900.**
     (c) If the apartment already exists, print the following error and continue with the program. **The program should detect this and respond immediately after reading the apartment number.** `ERROR: apartment already exists`
   - `R`: read an existing apartment's information
     (a) Prompt the user for an apartment number:
     `Enter an apartment number:`
     (b) Locate the specified apartment using the apartment number.
     (c) Print the apartment information in the following format:
     `Apartment Number:` *`apt_number`*
     `Tenant Name:` *`last_name, first_name`*
     `Lease Start:` *`lease_start`*
     `Lease End:` *`lease_end`*
     `Current Balance:` *`balance`*
     (d) If the apartment is not found, print the following error and continue with the program.
     `ERROR: apartment not found`
   - `U`: update an existing apartment record
     (a) Prompt the user for the following one at a time
        i. Apartment number **(zero-indexed integer)**
        ii. First name
        iii. Last name
        iv. Lease start date (string with slashes)

v. Lease end date (string with slashes)

vi. Balance (integer)

(b) Locate the specified apartment using the apartment number.

(c) Update each of the corresponding fields based on the user input. **If the user input is blank for a particular field (except apartment number), keep the original value from the file.**

(d) If the apartment record is not found, print the following error and continue with the program. **You should detect this and respond immediately after reading the apartment number.**
`ERROR: apartment not found`

- `D`: delete an existing apartment record

  (a) Prompt the user for an apartment number:
  `Enter an apartment number:`

  (b) Delete the specified apartment's data from **`apartments.dat`**

  (c) Print the following message to stdout with the apartment's number:
  `Record apt_number was successfully deleted.`

  (d) If the apartment is not found, print the following error instead and continue with the program.
  `ERROR: apartment not found`

  Hint: You may assume that the first name will never be an empty string. Use this to your advantage when identifying "deleted" content.

- If an invalid character is entered, print the following error and continue with the program.
  `ERROR: invalid option`

3. After an action is completed, display the menu again. This should go on indefinitely until `CTRL-D` or the end of a file is reached.

Locating Data

For the above functionality, **`apartments.dat`** should not be read sequentially to search for apartments. The location of the apartment in **`apartments.dat`** should be calculated immediately and directly accessed without performing a search.

Assignment Data

Input files for testing can be found in **`/usr/local/courses/rslavin/cs3423/Spring19/assign7`** including an existing .dat file and input for stdin. Copy these to your own assignment's directory.

Important: The input file assumes you are using the provided .dat file. Furthermore, each time you use the input file, you should refresh the .dat file to its original state.

Compiling Your Program

Your submission will include a makefile so the following command can be used to compile your code.

```
$ make assign7
```

Program Files

Your program should consist of up to three files:

- `assign7.c` - the main file which is compiled **(required)**
- `assign7.h` - an optional header file if necessary
- `makefile` - the makefile to make the `assign7` executable **(required)**

Verifying Your Program

Your program must work with the input provided in `a7Input.txt` and `apartments.dat`. To test it:

1. Place `apartments.dat` in the same directory as your compiled program.
2. Execute your compiled program and **redirect** `a7Input.txt` into it. You should not be copying or typing the contents of `a7Input.txt` into your terminal. Redirection must work.
3. Verify that the output is correct based on the input commands.
4. Execute your program again and use your program's menu to test that the information was correctly written to `apartments.dat`.

Submission

Turn your assignment in via Blackboard. Your zip file, named `LastnameFirstname.zip` should contain only your `makefile`, `assign7.c`, and possibly `assign7.h`. Do not include a .dat file.