Assignment 4: File I/O

CS 3424 - Systems Programming — Sam Silvestro - UTSA

For this assignment, you will use **C**'s I/O functions to create a simple course catalog database for administrators to update the details of each course offered by the CS department. The system will store basic information about each course, allowing the user to create, read, update, and delete them. All information for all courses 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 standard C library functions. Only binary I/O functions should be used to record data to the data file (it is permissible and expected, however, to use other C functions when prompting for user input).

Note: While this assignment is similar to Assignment 1, it is *not exactly the same*. You should thoroughly read all of these instructions. Lastly, you do not need to validate user input. That is, you can assume that all user input will comport to the specifications contained herein.

Storing Course Information

All course information will be stored in a single binary structure as records of the following structure, where **`course_Name`** is the name of the course (which may contain spaces), **`course_Sched`** represents the course schedule (either strings `MWF` or `TR`), **`course_Hours`** is the number of credit hours for the course, and **`course_Size`** is the number of students currently enrolled in the course.

```
1 typedef struct {
2     char course_Name[84];
3     char course_Sched[4];
4     unsigned course_Size;
5     unsigned course_Hours;
6 } COURSE;
```

The program will store all courses using the above `struct` in a single file called **`courses.dat`**, located within the same directory as the program (this file is provided to you). All courses will be referenced using their course number as their index (e.g., course ***i*** will be located in the data file at relative byte off ***i*** `* sizeof(COURSE)`. Course records will be stored in **`courses.dat`** in course-number order. Note that the course numbers will be specified by the user when a course is entered, and will not necessarily be sequential. If **`courses.dat`** does not exist, it should be created by the program ***in the current directory***. You must use the files located at:
`/usr/local/courses/ssilvestro/cs3424/Spring23/assign4`; specifically, **`courses.dat`** shall be used as the starting point for your database (i.e., you should use this file for all operations; however, your program *should* be capable of creating a new/empty data file when none is present in the program directory).

Program Execution

When the program is executed, the following actions should occur. **All program output should appear exactly as it appears below.**

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 course record
   U - update an existing course record
   R - read an existing course record
   D - delete an existing course record
   ```

2. The user then enters a one-character action (either upper or lowercase), leading to one of the following.

   - `C`: a course is created

     (a) From the terminal, read the following one line at a time:

     i. `Course number:` (should be a **zero-indexed** integer)

     ii. `Course name:` (string containing whitespace)

     iii. `Course schedule:` (string $\in \{\text{MWF}, \text{TR}\}$)

     iv. `Course credit hours:` (unsigned integer)

     v. `Course enrollment:` (unsigned integer)

     (b) Using the values entered by the user, update the corresponding course record in the `courses.dat` file.

     (c) If the course record already exists, continue to prompt for all remaining input, and only *then* print the following error before continuing with the program. That is, you should wait to detect this condition until after all input is gathered, *not* immediately after reading the course number.

     ```
     ERROR: course already exists
     ```

   - `U`: update an existing course record

     (a) Prompt the user for the following one at a time (you can expect the user input to be of the type and variety noted in the parenthetical adjacent to each prompt below):

     i. `Course number:` (should be a **zero-indexed** integer)

     ii. `Course name:` (string containing whitespace)

     iii. `Course schedule:` (string $\in \{\text{MWF}, \text{TR}\}$)

     iv. `Course credit hours:` (unsigned integer)

v. `Course enrollment:` (unsigned integer)

(b) Update each of the corresponding fields for the course based on the user's input. **If the user input is blank for a particular field (except course number, which is required and cannot be changed, as it is not stored as data in the file), maintain the original value from the file.**

(c) If the course record is not found, continue to prompt for all remaining input, and only *then* print the following error before continuing with the program. That is, you should wait to detect this condition until after all input is gathered, *not* immediately after reading the course number.
`ERROR: course not found`

- `R`: read an existing course's information

  (a) Prompt the user for a course number: (e.g., "3424")
  `Enter a CS course number:`

  (b) Search for the specified course using the provided course number (e.g., "3424").

  (c) Print the course information in the following format:
  `Course number:` *`course number`*
  `Course name:` *`course_Name`*
  `Scheduled days:` *`course_Shed`*
  `Credit hours:` *`course_Hours`*
  `Enrolled Students:` *`course_Size`*

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

- `D`: delete an existing course

  (a) Prompt the user for a course number (e.g., "3424"):
  `Enter a course number:`

  (b) Delete the specified course's record.
  Hint: You may assume the `creditHours` field will never be zero for a valid course.

  (c) Print the following message to standard output with the course's number:
  `course number` `was successfully deleted.`

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

- 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 proceed indefinitely until `CTRL-D` is read, or the end of the file is reached.

Locating Data

For the above functionality, **courses.dat** should not be read sequentially to search for courses. The location of the course in **courses.dat** should be calculated and directly accessed without performing a search (i.e., no looping!). Lastly, make use of the **dumprecs** command that I have provided in the course directory for this assignment. Grading will be performed by taking a random data file, running **dumprecs** with output redirected to a text file, and comparing your output to the expected output.

Assignment Data

Input files for testing as well as **dumprecs** can be found in **/usr/local/courses/ssilvestro/cs3424/Spring23/assign4** including an existing `courses.dat` file and input for stdin. Copy these to your own assignment's directory.

Important: The input file assumes you are using the provided `courses.dat` file. Furthermore, each time you use the input file, you should refresh the `courses.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 assign4
```

Program Files

Your program should consist of up to three files:

- **assign4.c** - the main file which is compiled **(required)**
- **assign4.h** - an optional header file, if necessary
- **Makefile** - the Makefile to make the **assign4** executable **(required)**

Verifying Your Program

At a minimum, your program must work with the input provided in **a4Input.txt** and **courses.dat**. To test it:

1. Place **courses.dat** in the same directory as your compiled program.
2. Execute your compiled program and **redirect a4Input.txt** into it. You should not be copying or typing the contents of **a4Input.txt** into your terminal. Redirection must work.
3. Verify that the changes made to the data file 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 **courses.dat**.

Note: this constitutes a bare-minimum function test of your assignment, and your code is expected to work for *any* combination of validly-formatted data files and input command sequences. These files will *not* be utilized in the grading of your assignment.

Submission

Turn in your assignment via Blackboard. Your zip file, named `a4-abc123.zip` should contain only your `Makefile`, `assign4.c`, and possibly `assign4.h`.

Do not include any data (i.e., `.dat`) files with your submission.