

ASSIGNMENT 7: FILE I/O

CS3423 - Systems Programming

Rocky Slavin & Sam Silvestror - 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 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).

Note: While this assignment is similar to Assignment 1, it is *not exactly the same*. You should thoroughly read all of these instructions.

Storing Course Information

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

```
1 typedef struct
2 {
3     char  courseName[64];
4     char  courseSched[4];
5     unsigned int  courseHours;
6     unsigned int  courseSize;
7 } 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/cs3423/Fall19/assign7; specifically, **data/courses.dat** shall be used as the starting point for your database (i.e., you must use this file for all operations, not a blank/empty/zero-byte data file, nor any other file).

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`
`R - read an existing course record`
`U - update 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 (**zero-indexed** integer)
 - ii. Course name (string possibly containing whitespace)
 - iii. Course schedule (string $\in \{MWF, TR\}$)
 - iv. Course credit hours (unsigned integer)
 - v. Course enrollment (unsigned integer)
 - (b) Using the values entered by the user, create a new entry in the data file based on the instructions above.
 - (c) If the course already exists, print the following error and continue with the program.
The program should detect this and respond immediately after reading the course number.
`ERROR: course already exists`
 - R: read an existing course's information
 - (a) Prompt the user for a course number: (e.g., "3423")
`Enter a CS course number:`
 - (b) Search for the specified course using the provided course number (e.g., "3423").
 - (c) Print the course information in the following format:
`Course number: course number`
`Course name: courseName`
`Scheduled days: courseShed`
`Credit hours: courseHours`
`Enrolled Students: courseSize`
 - (d) If the course is not found, print the following error instead and continue with the program.

ERROR: course not found

- U: update an existing course record
 - (a) Prompt the user for the following one at a time:
 - i. Course number (**zero-indexed** integer)
 - ii. Course name (string possibly containing whitespace)
 - iii. Course schedule (string $\in \{\text{MWF, TR}\}$)
 - 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), maintain the original value from the file.**
 - (c) If the course record is not found, print the following error and continue with the program. **You should detect this and respond immediately after reading the course number.**

ERROR: course not found
 - D: delete an existing course
 - (a) Prompt the user for a course number (e.g., "3423"):

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 immediately and directly accessed without performing a search (i.e., **no looping!**).

Assignment Data

Input files for testing can be found in `/usr/local/courses/ssilvestro/cs3423/Fall19/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 `courses.dat`. To test it:

1. Place `courses.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 `courses.dat`.

Submission

Turn in your assignment via Blackboard. Your zip file, named `abc123.zip` should contain only your `makefile`, `assign7.c`, and possibly `assign7.h`. Do not include a `.dat` file.