Computer Science 250

Project 3 – A Simple Database

**Due**: Mon. Apr. 15, at the beginning of class

Write a program that serves as a simple database system for college classes. The program will store the classes' department, number, and location. Initially, all the database information will be stored in a file. The user can perform four actions on the database:

1. List all of the classes in the database.
2. Add a class to the database.
3. Delete a class from the database.
4. Exit the program and write the current list of classes back to the file in the correct format.

The name of the file will be specified as the first and only command-line argument. The first line of the file contains an int value specifying the number of classes in the file. The rest of the file will contain the class information, one class per line. Here is the structure of the file:

<number of classes in this file>

<department name of class 1>:<number of class 1>:<location of class 1>

<department name of class 2>:<number of class 2>:<location of class 2>

...

You should assume that no piece of information in the file contains a colon: the colons are only used as separators. Any other character is legal in any field in the file.

When your program starts, read all of the class information in the file into an array of structs. Then, repeatedly prompt the user for an action, and perform the action as described below. Here are the details of the steps your program must perform.

**On startup**: Read all of the information in the file into an array of structs. You may assume that your program will never have to store information about more than 100 classes. You must dynamically allocate the memory for each struct, so the array will actually hold pointers to structs.

Action 1: The user will enter the string "list" at the prompt. To perform this action, iterate through the structs and print out each class, formatted neatly:

1: CS 100 - VH 1235

2: HIST 549G - OP 12a

...

Action 2: The user will enter "add" at the prompt. Your program should prompt the user for department, class number, and location, and than add a new struct containing this information to the array.

Action 3: The user will enter "remove" followed by a string. You program should remove the first class in the array that contains the entered string in either the department **or** course number field. Use the function strstr() from string.h to search for the class. If a matching class isn't found, do nothing.

If your program does find a match, don't forget to free the memory occupied by the struct that is being removed and then to compact the array.

Action 4: The user will enter "exit". Write the number of classes in the database, followed by all the information in the array, back to the file so that the file can be read the next time the program is run.

Here is the struct definition you must use to store each snippet.

#define MAX\_DEPARTMENT\_LENGTH 6

#define MAX\_COURSE\_NUMBER\_LENGTH 5

#define MAX\_COURSE\_LOCATION\_LENGTH 12

struct class\_info {

 char department [MAX\_DEPARTMENT\_LENGTH];

 char number [MAX\_COURSE\_NUMBER\_LENGTH];

 char location [MAX\_COURSE\_LOCATION\_LENGTH];

}

As always, use good programming practices when writing your program: separate your code into modules so that each module performs tasks related to a single part of the program, avoid magic numbers, use meaningful identifier names, etc.

**What to turn in**: When you are ready to turn the program in, print out a hardcopy of your source files. Make sure the main module is on top.

Then, create a tar file containing all of your source (and only your source) and a Makefile. The Makefile should have rules to build all object files, the executable, and rules for "all", "clean", and "tarball". When your tarball is unpacked, the command "make db" should build your executable, which must be called "db". Email me a copy of your tarball. Make sure the subject line of your email is "CS 250 – Section <#> - Project 3 <Last, First>".