Computer Science 250

Project 2: A Simple Database

**Due**: Tue. Mar. 23, at the beginning of class

Write a C program that serves as a simple database system for college classes. The program will store the academic department, class number, and location of classes, and allow the user to add classes to the list, and search the list of classes. Initially, all of the database information will be located in a file.

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 that are contained 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 can make the following assumptions about the data:

* No piece of data about a class contains a colon. The colon is only used as a separator between pieces of information about a class.
* There will be maximum of 64 classes in the file.
* Department names and class numbers will contain at most 7 characters. The class location will contain at most 31 characters.

When your program starts, read all of the class information in the file into an array of *dynamically allocated* structs. Do not allocate any more structs than are necessary. Once you have done this, repeatedly prompt the user for an action to perform.

There are four actions that the user can perform on the database:

1. List all of the classes in the database.
2. Add a class to the database.
3. Search for and print out the information about each class in the database that matches a particular substring.
4. Quit the program.

Here are the details of what your program should do for each action:

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, in a neatly formatted list, similar to this:

1: CS 100 meets in VH 1235

2: HIST 549G meets in 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 end of the array. Have the user enter each piece of information on a separate line, as shown below (user input in bold).

Enter department: **LING**

Enter class number: **275**

Enter location: **North Quad (BD 203 if raining)**

Action 3: The user will enter "search" at the prompt. Your program should then prompt the user for a string. Iterate through all the classes in the database, printing out information about each class where any part of the class (department, number, or location) contains the search string.

Action 4: The user will enter "exit". Terminate the program.

**Program structure**: When you are writing your code, you must separate the different parts of your source code into at least three different modules. Each module should handle a particular set of tasks in your program:

**The main module:** Contains the main logic of your program. Main() calls functions in other modules to implement the capabilities of your program.

**Add**: Contains one of more functions that query the user about a new class, and adds the class to the database.

**Search:** Contains one of more functions that query the user for a search string, and then iterate and print the appropriate classes.

Each module except the main should have an appropriate header file that gives the declarations of the functions that it contains.

Here is the struct definition you must use to store each class. You will need to store these definitions in an additional header file that can be included by the modules of your program that need them.

#define MAX\_DEPARTMENT\_NAME\_LENGTH 8

#define MAX\_COURSE\_NUMBER\_LENGTH 8

#define MAX\_COURSE\_LOCATION\_LENGTH 32

#define MAX\_COURSES 64

// Information about one class

typedef struct class\_info {

char department [MAX\_DEPARTMENT\_LENGTH];

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

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

} class\_info\_t;

// Array of classes

class\_info\_t \*course[MAX\_COURSES];

// Number of classes in the array

int n\_courses;

Finally, you must create a Makefile that can be used to build your program. You Makefile must contain rules to build each module, another rule to build the executable, and rules to make all, clean, and tarball.

As always, use good programming practices when writing your program: Use meaningful identifier names, limit the use of global variables, avoid magic numbers, break complex tasks in to functions, 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. Include a copy of your make file.

Then, create a tar file containing all of your source (and only your source) and the Makefile. Upload your tarball to blackboard.