Computer Science 250

Project 3 – A Simple Database

**Due**: Wednesday, Nov. 14, at the beginning of class

Write a program that serves as a simple database system. Your program will manipulate information about people. For each person, the program will store the person's name and address, which we will call a "snippet". The user can perform four actions on the database:

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

Initially, the information in the database will be contained in a text file. The name of this file will be specified as the first and only command-line argument. The first line of the file will contain an int value specifying the number of snippets in the file. The rest of the file will contain the snippets, one per line, in the following form:

<# snippets>

<name 1>:<street address 1>:<city 1>:<state 1>:<zip 1>

<name 2>:<street address 2>:<city 2>:<state 2>:<zip 2>

...

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

When your program starts, it should read all of the snippets in the file into an array of structs. It should then repeatedly prompt the user for an action, and then perform the action according to the specifications described below. Here are the details of the steps your program must perform.

**On startup**: Read all of the snippets in the file into set of structs. You must dynamically allocate the memory for each struct. Store the pointers to the structs in an array. You can assume that your program will never have to deal with more than 100 snippets.

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

Ann Archer

123 Ace Ave.

Ames, AL 12345

Action 2: The user will enter "add" at the prompt. Your program should prompt the user for name, address, city, state and zip, and add a new struct to the array.

Action 3: The user will enter "remove" followed by a string. You program should remove every snippet from the database that contains the entered string as a substring. (Hint: use the function strstr() from string.h.) Don't forget to free the memory occupied by the structs.

Action 4: The user will enter "exit". Write the number of snippets followed by each current snippet back to the file.

You can assume that you will never need to store more than 100 snippets in memory when you declare the array.

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

#define MAX\_NAME\_LENGTH 32

#define MAX\_ADDRESS\_LENGTH 64

#define MAX\_CITY\_LENGTH 16

#define STATE\_LENGTH 3

struct snippet {

char name [MAX\_NAME\_LENGTH];

char addr [MAX\_ADDRESSS\_LENGTH];

char city [MAX\_CITY\_LENGTH];

char state [STATE\_LENGTH];

unsigned int zip;

}

As always, use good programming practices when writing your programs: separate your code into modules, 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>".