Computer Science 250

Project 1: Working with Files and Strings in C

**Due:** Tue. Feb. 16, at the beginning of class

**Reminders**

* You are not allowed to work with, consult, or seek help from anyone except the tutor or instructor as you plan, write, and debug your program.
* You must use the computer science department server ice, along with the clang compiler and the emacs editor installed on ice to develop your program. Neither the tutor nor I will give any assistance in developing or debugging unless you are using this programming environment.

A file contains a list of student names, majors, and G.P.A.s. You are to write a program that will read the contents of this file into an array of structs. The name of the file will be given on the command line. Your program should repeatedly prompt the user for a string and print out all of the information about the first student found whose name contains that given string as a substring.

Information about each student will be given on three consecutive lines. For example, here is the contents of a file that contains information on two students:

Dennis Ritchie

CS

4.0

Amanda B. Recondwith

ART

2.5

Here is an example of what you program should look like when it is executed with the text file “students.txt” as the data file. (The ‘%’ is the shell prompt.) User input is shown in bold.

% **./project1 students.txt**

Please enter part of a student name: **Ritch**

Name: Dennis Ritchie

Major: CS

G.P.A.: 4.0

Please enter part of a student name: **foo**

No student found

Please enter part of a student name: **o**

Name: Amanda B. Recondwith

Major: ART

G.P.A.: 2.5

Please enter part of a student name: **<ctrl-d>**

You can make the following assumptions about the data in the file:

* The name of the student will have no more than 32 characters.
* The G.P.A. will have no more than 4 characters.
* The G.P.A. will be a properly formatted float.
* Every line of the file will have the correct format – there will be no blank or otherwise malformed lines in the file.
* There is information on, at most, 100 students in the file. However, they may be fewer than 100 students.

You should store each class and its location in an array of structs. Use the following definitions and declarations at the beginning of your source file:

#define MAX\_STUDENT\_NAME\_LEN 34

#define MAX\_MAJOR\_NAME\_LEN 6

#define MAX\_STUDENTS 100

typedef struct student\_info {

 char name[MAX\_STUDENT\_NAME\_LEN];

 char major[MAX\_MAJOR\_NAME\_LEN];

 float gpa;

} student\_info\_t;

Use good programming practices when writing your code. Points will be deducted for poorly written code, even if it executes properly. I will be paying particular attention to the following things when your source code is graded:

* Use of good identifier names, properly written using C conventions: snake\_case for all identifiers except constants, which should be in SNAKE\_CASE.
* Use constants for any literal values that have a special meaning in the program. No magic numbers!
* Employ a good commenting style: include a “header” comment at the beginning of the program that briefly describes what the program does, along with the author, and the date. Make sure each function has a short comment explaining what it does, what parameters it takes, and what value, if any, it returns.
* Use proper indentation.
* Make proper use of functions – break the main program into simpler tasks, and perform those tasks in functions.
* Don’t reinvent the wheel when your program needs to manipulate C-strings. Make sure that you are familiar with all the functions that the string library provides (see appendix B.3 of the text) before you begin to code.

**How your code will be evaluated**: The grader will be run on a series of top-secret test cases involving different input files and different sets of requests. Points will be deducted for each test case where your program crashes, hangs, or gives incorrect output. In addition, points will be deducted for poorly written code (see above).

Make sure that you thoroughly test your program before you submit it. Use emacs to create several text files of the proper format with varying types of data. Then, run your program several times on each file.

**What to turn in**: When you are ready to submit your program, print out a hard copy of the source file to submit in class. Then, upload a copy of your source file to the appropriate place in blackboard.