McGill University
School of Computer Science
COMP206 Software Systems

Assignment #3

Due: March 11, 2014 at 23:55 on My Courses

C Programming

QUESTION 1 – MENU PROGRAM (data set array manipulation & redirection)

Purpose: Every application needs a good user interface. In this question we will explore writing a good text-based user interface. We will also practice manipulating arrays.

Application area: Many scientific and engineering problems contain a lot of data that need to be processed and organized properly in a report.

Program: Create a C program called DataSet.c. Compile this file into the executable file name dataset. To launch the program from the command-line prompt you must enter:

$ ./dataset < filename

Where 'filename' is the name of a text file that contains all the data from a series of experiments. The data is formatted in a specific way. First, you do not know beforehand how many separate lines of input exist in this file. You do know that the very last line of input file contains the word *** END *** and that the maximum number of experiments can never exceed 20. Second, you know that the experiment data is organized in pairs. The first line is the name of the experiment and the second line contains exactly 10 numbers representing the data acquired for that experiment. For example:

Experiment One
3 10 8 7 3 2 9 7 5 6
Experiment Two
10 20 30 40 50 60 70 80 90 100
Control Group
5 5 5 5 5 5 5 5 5 5
*** END ***

The program begins by accepting experimental data from 'filename' and storing that into arrays. After the input of the experimental data a user interface is displayed.

The program assumes that the experimental input is being entered using a command-line redirection (as shown above). It also assumes that the redirected file is formatted as described above. Your program does not need to do any validation for the redirection. Remember that redirection sends information to STDIN and STDOUT. In this case to STDIN. This means any of C's I/O programs that use STDIN can receive data from this

redirection. Since STDIN is the keyboard and the user types with that, it means that you can program imagining that a user is entering that information and that it is not originating from a file. The behavior would be similar.

The program should run gracefully. This means that if there is an error in the input of data the program does not crash. Instead it either prompts the user to try again or it makes assumptions and continues executing. This is important for the experiment data input using redirection. Since you are not directly entering that information by hand an error could occur and you would not be able to correct it by hand (the redirection makes the data entry automatic).

The program uses two arrays defined in the main function:

```
char *experiments[20];        // Stores the experiment names
int data[10][20];             // Max 20 experiments each with max 10 data points
```

The program uses the following global variable (no other global vars permitted):

```
int numExperiments;           // Calculated total number of experiments from file
```

As the redirected file is read in, numExperiments is incremented to represent the total number of experiments present in the file. In our example above, it would assign 3 to numExperiments. Using the above example, the array experiments[] would have the string “Experiment One” in cell 0, “Experiment Two” in cell 1, and “Control Group” in cell 3. Row 0 of the array data[] would contain the ten numbers from experiment one. Row 1 would contain the numbers from experiment one, and row 2 would contain the numbers from the control group.

Note: You cannot directly scanf into a char* array. You will have to first read it into a simple character array and then use the string.h library command strdup() to convert the contents to a char* string. Then you can assign the pointer to the experiments array.

The program will then display the following menu:

DATA SET ANALYSIS
1. Show all the data
2. Calculate the average for an experiment
3. Calculate the average across all experiments
4. Quit

Selection: __

The menu will be displayed using a while-loop until the user presses 4 to end the program. A switch-statement must be used to process the user's input with a default case if the user entered an incorrect menu selection. The prompt “Selection: “ is displayed with the cursor at the colon waiting for user input.

If the user selects option 1 then the switch-statement calls a function called displayAll(). You will need to determine what parameters must be passed and if anything is returned.

This function will display to the screen all the data stored in the two array. You can format the output as you see fit. Just make sure that it is easy to read.

If the user selects option 2, then the switch-statement will call a method called individualAverage(). You will need to determine what parameters must be passed and if anything is returned. Before this function is called, from within the switch-statement, prompt the user for the name of the experiment they want to use. Do not ask for the row number. The function individualAverage() will use the name to determine which experiment to use. The function then returns the average to the switch-statement. The average is then displayed from the switch-statement in a nice way.

If the user selects option 3, then the switch-statement will call a method called allAverage(). You will need to determine what parameters must be passed and if anything is returned. This function will automatically compute the average across all experiments returning the result to the switch-statement. The result is displayed from the switch-statement in a nice way.

What to submit and grading:

- Submit the .c file and a sample 'filename' file.
- This question is worth 7 points (assignment is worth total of 20 points)

QUESTION 2 – A CSV DATABASE (command-line inventory)

Purpose: Almost all applications need to read and write to files. This question explores file manipulation and string processing. String processing is another important programming skill.

Application area: Database programming. In our case, a subset of database programming called CSV files.

Program: A CSV file is a structured text file. It stands for Comma Separated Vector. A normal text file has no structure. In other words, a human can write characters into a text file in any way they like. A CSV file is not like that. It has rules. The first rule is that every row of a CSV file is a series of characters terminating with a carriage return and line feed. The row is called a "record". A record is responsible of keeping information about a single object. For example, a record could store information about a friend of yours. All the records in the CSV file would represent all of your friends, each friend one of those records. A record is subdivided into "fields". A field contains one fact about the record. For example, if the record is information about your friend then the fields would be the name, age and gender of that friend. In a CSV file a field is separated from another field by the comma. Therefore, you cannot use the comma, carriage return, and line feed as data since they have a special meaning in formatting the CSV file.

In this assignment question you will create a command-line driven inventory control program for a restaurant. As an example, here is a sample inventory CSV file. Each

record represents information about one food item the restaurant has in its inventory. If the store has 20 different food items then there will be 20 distinct records in the CSV file, one for each food item. Each record is divided into the same fields: the first field is the name of the food item, the second field represents the quantity of that food currently available, the third field represents a reorder limit. If the quantity currently available (currently in stock) goes below the reorder limit number then this is a situation that warrants a warning message since the store is almost out of this item. The last field is the unit cost for that food item. It will cost that much money to replace one unit of that food item. The example CSV file:

hotdog, 10, 2, 1.50
bun, 10, 2, 0.50
burger, 100, 10, 2.00

The above example deviates from the CSV definition by using the comma and space as the field separator. You may use this variation or the official definition in your CSV file.

Using Vi, create a text file called inventory.csv and input 10 food items following the format given to you above. In other words: food name, quantity in stock, reorder limit number, and unit cost in dollars and cents.

Now write a C program called Inventory.c and compile it to the executable name inv.

This will be a command-line driven program with the following syntax:

$ ./inv COMMAND ARGUMENTS

Where:

- The command-line program name is inv.
- The following commands are valid: list, reorder, deduct, add.
  - The command "list" does not use any arguments. At the command-line the user would write:

    $ ./inv list

    The program would display on the screen, in a nice way and using column headers, all the items (all records and fields) in the CSV file. The commas should not be part of the output. It is up to you how "pretty" the output should be.
  - The command "reorder" does not use any arguments. At the command-line the user would write:

    $ ./inv reorder

    The program would display nicely, using the same formatting as "list" but

with the title "MUST PURCHASE THE FOLLOWING ITEMS". Then a listing of only those food items where the quantity in stock is <= reorder limit.

- The command "deduct" uses one argument, the name of the food item. At the command-line the user would write:

  $ ./inv deduct bun

  The program would then find the bun record and subtract one from the quantity in stock.
- The command "add" uses four arguments, food name, quantity in stock, reorder limit number, and unit cost. The user would write the following at the command-line:

  $ ./inv add 7up 20 5 0.75

  The program will append a new record at the end of the file containing the above information.

This program has no menu. Once it finishes processing the command it returns to the command-line.

You will need to use the string.h file for string manipulation. You will also need to use stdio.h for the FILE commands. Arrays will also be useful. The CSV file is updated each time you run this command. The program always opens the inventory.csv file.

What to submit and grading:

- Submit the .c file and a sample inventory.csv file.
- This question is worth 8 points (assignment is worth total of 20 points)

QUESTION 3 – THE SYSTEM COMMAND (my shell)

Purpose: To communicate directly with the operating system.

Application area: Systems programmers often require the operating system to perform tasks for their program. Why write the code yourself when the operating system already knows how to do it? Or, the operating system's security features block you from doing some activity that the OS permits itself to do... so, ask the OS to do the activity for you.

Program: I've already asked you to do two big-ish programs, so the solution to this question is a small program. In the library file stdlib.h is the system() function. It permits you to send command-line commands to the operating system. It is formatted this way:

```
int system(char *string);
```

For example if you write: x = system("ls"); or just, system("ls"); then the computer will display on the screen the files from the current directory.

Write the following program called myshell.c which you must compile to the executable mysh. You will launch this program from the command-line by writing:

```
$ ./mysh
```

Your program will use a do-while-loop that loops until the user enters the word "done". The program prompts the user with the string ">> ". Then the program waits for the user to type in a string. If the string entered is not the word "done" it uses the string in the system() function call. This will send the string to the OS for execution. The results of the execution will be displayed on the screen. Then the program loops to the top, repeating the above steps until the user enters the word "done". When the program terminates it displays "Thank you for using mysh!".

What to submit and grading:

- Submit the .c file.
- This question is worth 5 points (assignment is worth total of 20 points)

HOW IT WILL BE GRADED

- Your program must run in order for it to be graded. If your program does not run the TA does not need to go any further and may give you zero.
- Your program must run on the Trottier computers since this is a programming course for the Unix environment. The TA will not modify your text files in any way to get them to run under Trottier's operating system.
- All grades are awarded proportionally. In other words, if a question is graded out of 4 points and you got 75% of the program running then you will get 3 out of 4, etc.
- Grading portions of your code that do not run (assuming that your program runs at least minimally) will be graded proportionally as to how correct it is.
- Make sure your code is easy to read since this may impact the quality of your grade.