Project 1: Database Management

CS 4350\_251

Group 6:

Roger Escobedo

Sarah Forbis

Denise Gan

Kemal Guddeta

**Section I: Introduction**

This report provides a summary of our group’s approach to the assigned Project 1. Group members collaborated on a shell script that manages a text-based database of contacts. Basic functions to add, update, remove, and find a contact record were required. Additional functions were designed to improve the usability of the database for a user. These additional functions, along with the database text file, were designed in a cooperative fashion by group members.

Roger Escobedo was responsible for writing Section III of the report and for implementing the menu, create\_database() and add\_record() functions. Denise Gan was responsible for writing Section 1 of the report as well as the find\_record() and list\_record() functions. Sarah Forbis wrote Section II of the report and wrote the update\_record() and sort\_records() functions. Kemal Guddeta wrote Section IV of the report and implemented the delete\_record() and export\_records() functions.

While the explicit responsibilities of each group member are detailed above, it should be noted that every group member contributed to the improvement of the entire script. The entire group participated in testing the shell script, troubleshooting and solving problematic functions, and adding documentation to clearly explain the workings of the script. It was important for the learning process for team members to make suggestions on how to better implement certain functions. Team members regularly corresponded via email or through in-person meetings. From start to finish, this project was a group effort that resulted in a successful shell script that meets the assignment’s objectives.

**Section II: Database Design**

The first line of the database is the header line, and marks each field: unique ID, first name, last name, street address, zipcode, phone, and email. The unique ID field is for easily searching, editing, or deleting a record within the database. It also provides a way to sort the database in numerical order for the user. The first and last name, street address, zipcode, phone, and email all what their names imply and store general information about the person in the record.

Each field is separated by a ‘:’ without any spaces except within the field itself. Take the example below:

UniqueID:FirstName:LastName:StreetAddress:Zipcode:Phone:Email  
1:John:Apple:1200 N LBJ DR:78666:5124561234:jApple@txstate.edu  
2:Qijun:Gu:206 Nueces:78666:5122453518:qijun@txstate.edu  
3:Ponyboy:Curtis:1965 Tulsa OK:43123:9564328888:pbcurtis@ou.edu  
4:Susan:Hinton:1948 Tulsa OK:98000:2348784222:shinton@utexas.edu

In the fourth field, which contains the address, there can still be spaces. Colons are only needed when separating each field from each other. When the first new record is added to the database, it is written below the header line. Every Time a new record is added after that, it is written below the record written before it. That is, each new record is written on the next line in the file.

**Section III: Function and Interface Design**

The create\_database function is included as an easy to read representation to create a database file. The function contains two conditional statements that creates a directory if the supplied directory does not exist, and the database file if it also does not exist as well as the instantiated header fields. It's functionality is not grandiose, but the effect of the function being in the menu logic allows for a better flow for anyone utilizing or reading the program with the function.

The find\_record function is required for any CRUD operation among any database program. The find\_function searches through a very specific pattern given by the user's input. The pattern itself is must match every character in the string between the colons, since it is a colon separated file. The pattern is case insensitive. When searching for a record, the query returns multiple results. The user might intentionally be wanting to find all related records . The records found will then be cleanly outputted to a display so the user can easily see the header fields aligned with the queried results.

The add\_record function is necessary again for any CRUD operation among any database program. The function checks whether or not the supplied implicit argument list contains a colon or not. This design feature was heavily needed in case an invalid character, such as a colon (:) was entered in as a record. If the condition was not checked, the file itself would allow for certain records to be unreadable. There are two cases for generating a unique id, the first case is when there is no record in the database in which the first entry will be given a unique id of 1, however any other record inserted after will be given a (unique id +1 ) relative to the most previous inserted record. Each record must be unique at least in one field, and the ID column ensures that uniqueness. This is common among database design. The generated ID relieves the user from having to worry about always creating a unique id for each entered in record.

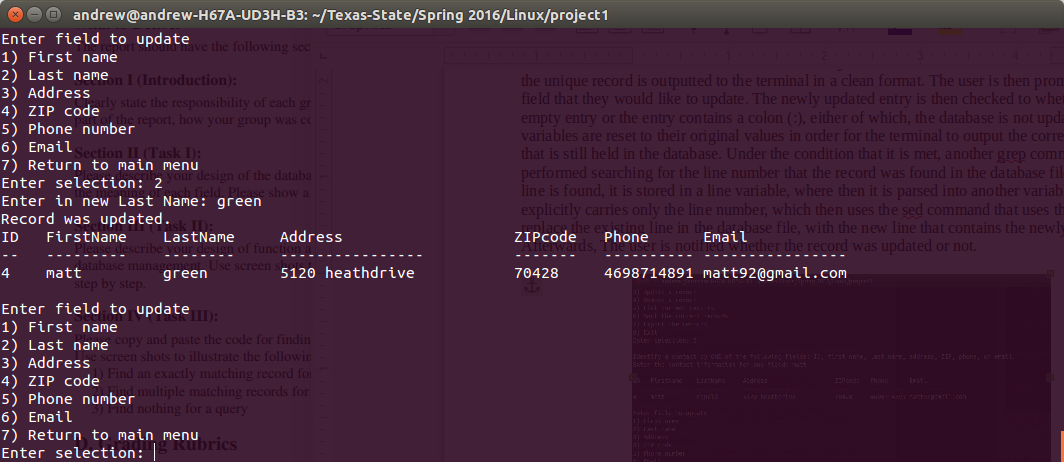
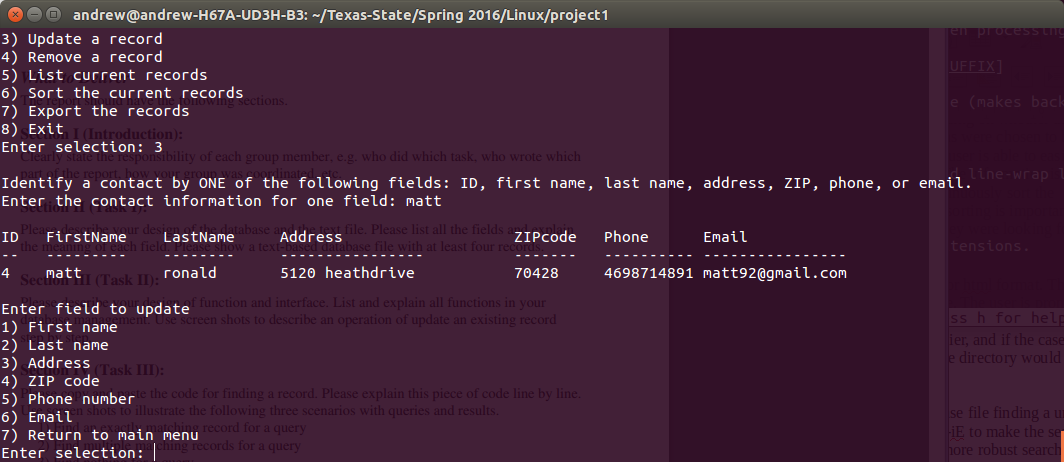
The update\_record is yet another CRUD operation in working with any database. The function finds the specific record matching the pattern the user had entered in, case insensitive. The case insensitive search was included to make the user's interaction with the program less of hassle in order to query the result with more ease. The user is notified that the search must be unique if the queried result finds more than one record, or that there are no records when no results are found. This design feature was implemented to provide the user feedback to better query their desired results. If the results of the query is found with one record, the user is given a formatted representation of the record with every field displayed on the screen. The user should not have to be responsible to remember each value for each field, so this implementation was crucial. The user should however not have access to the record's unique id, and so the user is never prompted to change that field to ensure uniqueness, but is prompted on every other field. After the user has decided what field they want to update, they are prompted to provide a new value, and then that value is checked to ensure that it is not an empty string. No entry in a record is allowed to be empty as well as contain a colon. The newly updated record is then inserted in it's original position as it was in the original database file. This decision was made so that each record is ordered numerically in the database file no matter which record was updated.

The delete\_record function is the last CRUD operation needed when manipulated a database file. The search command works the same as the other operations. Finding a length sensitive, but case insensitive search results. The function only operates on record at a time, meaning that only one record can be deleted even if multiple records match the user’s inputted pattern, however if there are multiple records that are a resultant from the query the user is notified that the record was unable to be removed. The reason for this is that the type of data that this database is carrying could be considered sensitive, and the record list, for its purpose, should be relatively small. Forcing the user to be more attentive towards the removal of each record, while also not creating too much of a burden for the user to manually remove each record.

The list\_records function lists all the records that are currently in the database. The functions use was convenient for debugging purposes, but also is valuable to the user, assuming that the database file should stay relatively small for its intents and purposes. The function cleanly outputs all of the data from the database into an easily readable format.

The sort\_record function sorts all the current records in the database depending on whether the user wants the records sorted by id, first name, last name, or email. These options were chosen to be sorted by these fields since they provide the most meaning to the user. The user is able to easily sift through each record alphabetically or numerically depending on the field. The information is the outputted to the terminal and formatted cleanly. The user is allowed to continuously sort the database until they are ready to return to the main menu. This continuous sorting is important for the case that the user may not have been able to find the information that they were looking for from their first sort.

The export\_record function easily exports the database into a text, word, or html format. The user is specified to enter in a directory to store the desired format of the database. The user is prompted if the directory does not exist, and the function tries to create the directory. The automation of creating the directory made the users time with the function much more easier, and if the case came about that the directory was wrongly named, then the manual deletion of the directory would not prove overly cumbersome.

The update function grabs the implicit given argument and greps the database file finding a unique record that matches the given pattern. The grep command uses the options -iE to make the search case insensitive and using regex to be more readable as well as provide a more robust search which then redirects the output to a temp\_file. 7 variables are created to hold each value corresponding to each colon separated field in temp\_file. The temp\_file is then checked on whether or not there was either more than one record found, or none. If only one record is found, the condition is met, and the unique record is outputted to the terminal in a clean format. The user is then prompted on which field that they would like to update. The newly updated entry is then checked to whether it is an empty entry or the entry contains a colon (:), either of which, the database is not updated, and the variables are reset to their original values in order for the terminal to output the correct information that is still held in the database. Under the condition that it is met, another grep command is performed searching for the line number that the record was found in the database file. Once the line is found, it is stored in a line variable, where then it is parsed into another variable that explicitly carries only the line number, which then uses the sed command with the -i option to replace the existing line in the database file, with the new line that contains the newly updated entry. Afterwards, The user is notified whether the record was updated or not.

**Section IV: Scenarios with Queries and Results**

This section explains codes of find\_record function in detail and illustrates the function results in different scenarios with queries. Following is a copy of the code of find\_record function used for this project. But in this report for the sake of convenience of explaining the code line number is added.

1 Purpose: Finds record in contact database based on user search term

2 function find\_record(){

3 #counts number of entries containing search term found in database

4 count=$(grep -icE "^$1:|:$1:|:$1\$" $FILE)

5

6 if [ $count == 0 ]; then

7 echo "Search term not found."

8 else

9 #finding all entries containing search term in the database and storing in results\_file

10 grep -iE "^$1:|:$1:|:$1\$" $FILE > results\_file

11 #Note: Displayed results may be truncated if record information exceeds set limit

12 awk -F ":" '

13 BEGIN { format="%-4.4s %-12.12s %-12.12s %-25.25s %-9.9s %-10.10s %-25.25s\n"

14 printf format, "ID", "FirstName","LastName", "Address", "ZIPcode", "Phone","Email"

15 printf format, "--", "---------", "--------", "----------------", "----------", "----------","----------------" }

16 {printf "%-4.4s %-12.12s %-12.12s %-25.25s %-9.9s %-10.10s %-25.25s\n", $1, $2, $3, $4, $5, $6, $7}

17 ' results\_file

18

19 #gives user option to save results

20 read -p "Keep and save results as 'results\_file.txt'? Y/N? " select

21 case $select in

22 N|n)

23 rm -rf results\_file

24 echo "File not kept.";;

25 Y|y) echo "File kept and saved.";;

26 \*) echo "Invalid entry.";;

27 esac

28 fi

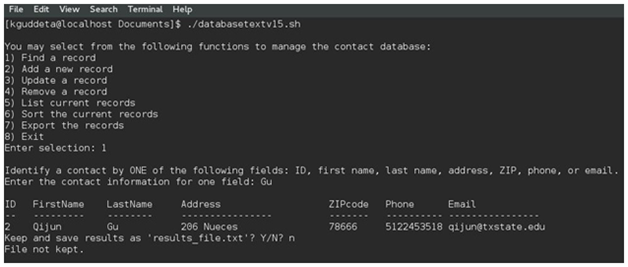
29 }

The 4th line in the find\_record function counts the number of entries containing search term in the database and assigns the value to count variable. If the value of count is equal to zero a message “Search term is not found “is displayed on terminal while if it is different from zero blocks of statements under else will be executed. Lines 12 to 17 reads the contents of result\_file and generates a formatted report specifying the input field separator to “:” . Line 14 and 15 print headers for the find query to be displayed. Line 20 gives user an option to save results by prompting a user to enter either Y for agreement and N for disagreement. If a user selects either N or n then commands in line 23 and 24 are executed. Line 23 removes results\_file while command in line 24 displays a message “File not kept.” However, if a user's selection is either Y or y a command in line 25 will be executed and a message “File kept and saved” will be displayed. Any other selections by user’s makes a command in line 26 executed and a message “Invalid entry.”will be displayed.

Based on the value of count which is the number of entries in the database file containing the search term three scenarios are explained as follows. The first one is a situation where there is exactly one matching record while the second is a case where multiple matching records are found. The third case is where there is no matching entry containing the search term.

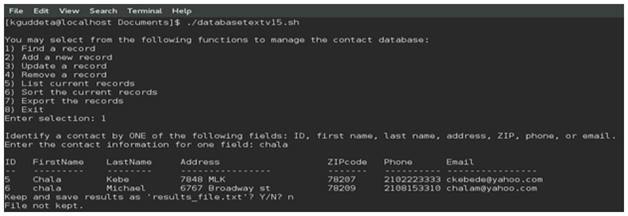
**a)** **Find an exactly matching record for a query**

Following screenshot displays a result when “Gu” as a search term. Since we have only one entry with Gu as a last name in the database only a record entry containing “Gu” which is stored in result\_file and thereby accessed by commands in line from 12-17 to display a formatted record.



**b)** **Find multiple matching records for a query**

To illustrate the case where multiple entries containing two records with same first name were added to the database. Since the value of count is 2 (different from zero) blocks of statements under else are executed. When a statement in line 10 is executed, all the entries with search term (which are entries with ID number 5 and 6) are stored in the result\_file. Then command in line from 12-17 displays the records in result\_file (which are records in with ID number 5 and 6).



**c)** **Find nothing for a query**

If the value of count is zero (there is no search term in the database), line 7 is executed and a message “Search term is not found “will be displayed on terminal. The next screenshot is taken where a user enters a search term ‘David’ which is not in the database.

