Group 13 Project 1

Nicholas Garcia, Anastacio Monsivais, and Brandon Rose

Cover Sheet

Section I (Introduction):

Group 13 is comprised of Nicholas Garcia, Anastacio Monsivais, and Brandon Rose. As we came together we initially designated specific function to each group member but eventually we went to the library and put our laptops to a large screen TV and planned and coded as a team. Nicholas took the lead on the main menu, the add function and the main body of the program while Anastacio took the lead on the search function that was used in the update and remove functions. Brandon took the lead on the database functionality and general proofreading all parts of the code allowing the rest of the group to lean on his knowledge of shell programming on their responsibilities. As far as communication and sharing of code goes, the group communicated via text messaging on our mobile phones for planning the meetings, and then implemented google drive for distribution of the code. Figure 1 shows the files on the local machine which stores the files while figure 2 shows these files and who they are shared with from the google website. The report was written as a group while displayed on the monitor with each group member giving input while Nicholas was at the keyboard. Each member contributed to the project in all components including writing of code and of the report.

Figure 1

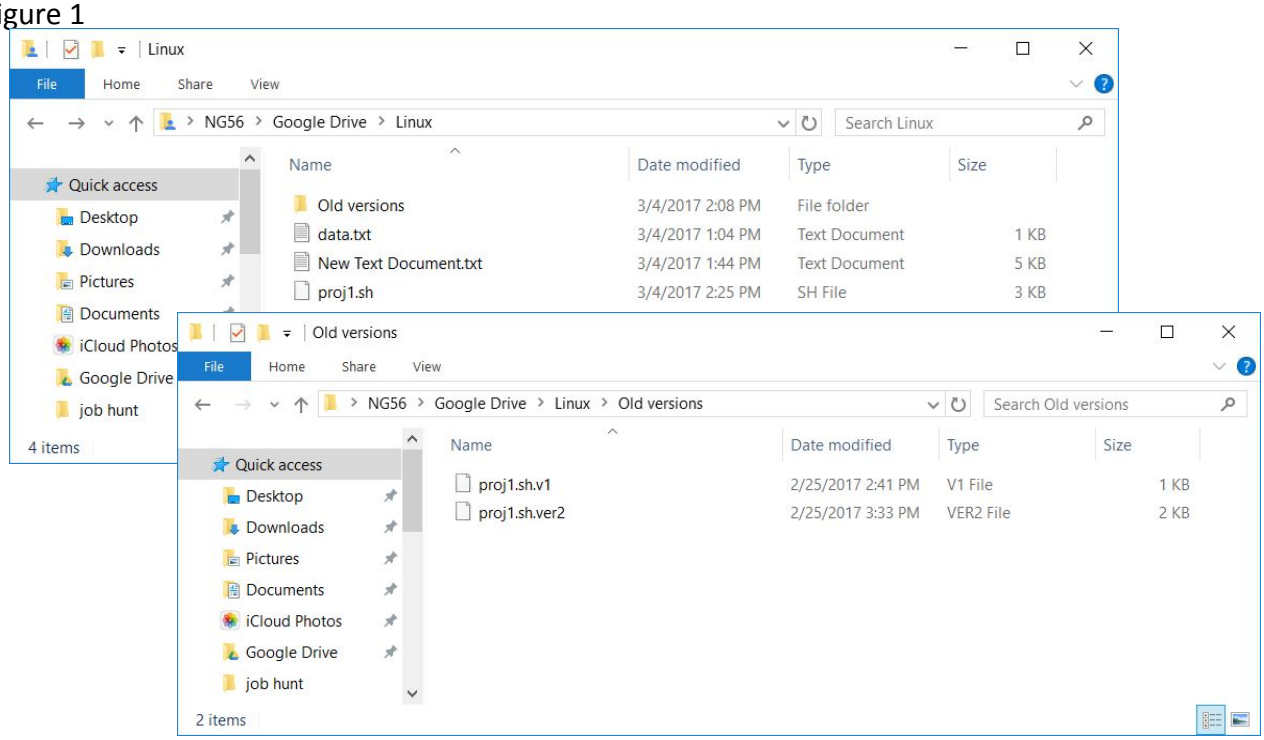
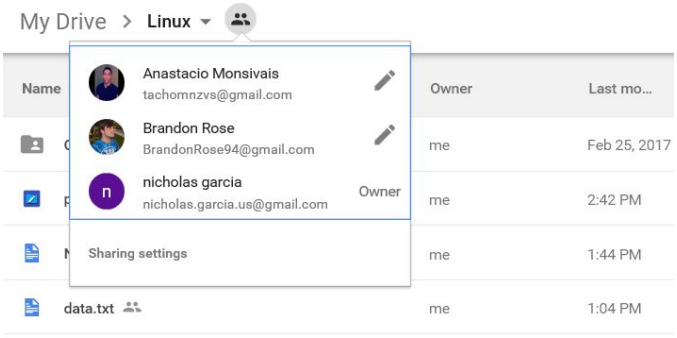


Figure 2



Section II (Task I)

The design of the database that we implemented was a two files system. We have our “real” file which was named “data.txt” and a “phantom database” which we would use to manipulate the data. Once the data was manipulated and we could trust it, we would then make the phantom database the new real database. This would help to ensure that no matter what happens during the program mid execution the database would be shielded from being corrupted. Our format was created with using semicolon as the delimiter/separator. We removed the titles from the database file as we found it was not needed and showed up as an extra entry.

Contact name; this entry is used for the user to input the name of the contact. There are no limitations for characters or size.

Example: Charlie Chuckles

Contact address; This entry is used for the address of the contact. There are no limitations for characters or size.

Example: 789 North Street

Phone Number; This entry hold the phone number for the contact. There are no limitations for characters or size.

Example: 5126543217

Email Address; This entry is used to hold the contact’s email address. There are no limitations for characters or size.

Example: CC@gmail.com

raw data:

Qijun Gu;206 Nueces;2453518;qijun@txstate.edu
John Smith;123 West Street;5121234567;JSmith@yahoo.com
Bobby Bibbler;456 East Street;5127894561;BB@icloud.com
Charlie Chuckles;789 North Street;5126543217;CC@gmail.com

formatted data (ease of viewing):

Qijun Gu	206 Nueces	2453518	qijun@txstate.edu
John Smith	123 West Street	5121234567	JSmith@yahoo.com
Bobby Bibbler	456 East Street	5127894561	BB@icloud.com
Charlie Chuckles	789 North Street	5126543217	CC@gmail.com

How we formatted the data for display in our program:

Name: Qijun Gu Address: 206 Nueces Phone: 2453518 Email: qijun@txstate.edu
Name: John Smith Address: 123 West Street Phone: 5121234567 Email: JSmith@yahoo.com
Name: Bobby Bibbler Address: 456 East Street Phone: 5127894561 Email: BB@icloud.com

Name: Charlie Chuckles Address: 789 North Street: 5126543217 Email: CC@gmail.com

Section III (Task II):

Functions:

DisplayMenu ()
Description: show menu option

openAndCopy()
# Description: This function checks to make sure that the database exists and that a local copy has been made for use by other functions

replaceWithUpdate()
# Description: This function is to replace the data.txt with the updated .data.txt. This is done to make sure that the data.txt does not become corrupted by the program mid stream

Add ()
Description: Add entries to the address book and then executing the replaceWithUpdate

List ()
Description: List entries of the address book, with correct formatting

Find()
#Description:Finds the line number and prints the line containing any input character entered by the user

removeRecord()
#Description: removes a record by removing the line from the file and writing the desired lines to the local file, see figure 3 and its explanation below for more clarification

updateRecord()
#Description: works with the combined efforts of removeRecord and Add

Figure 3

```
#updateRecord
#Description: removes a record by removing the line from the file
updateRecord() {
    if Find; then
        echo -n "Enter a line to update or 0 if none: "
        read rmLine
        if [ "$rmLine" -eq 0 ]; then
            return 0
        fi
        remTempFile=".rem.txt"
        touch "$remTempFile"
        counter=1
        while read -r name address phone email
        do
            if [ "$counter" -ne "$rmLine" ]; then
                printf "%s$IFS" $name >> "$remTempFile"
                printf "%s$IFS" $address >> "$remTempFile"
                printf "%s$IFS" $phone >> "$remTempFile"
                printf "%s\n" $email >> "$remTempFile"
            fi
            counter=$((counter + 1))
        done < "$localFile"
        cp "$remTempFile" "$localFile"
        rm "$remTempFile"
        replaceWithUpdate
        Add
    else
        return 1
    fi

}
```

As in in figure 3, our updateRecord functions works with the principle of writing the information we want to keep, and then excluding the information we don't. This information then is put onto our master file after working through an intermediate file. The first step is to find the record to update. After that we set a counter equal to the record number we want to remove. The loop will write the data for any line that is not selected, while skipping the record line. This is done the increments counter. Once complete the system will write the data to the local database, then update the master with the local (replaceWithUpdate function) then prompt the user to add a new record that is "replacing" the original (appends to the bottom through the add function).

Issues with our code that we are aware of would be some kind of input validation. For example if a contacts information has a ";" anywhere in their record it will mess with the system, also if the user selects a record number outside of what was search, the user number will be deleted, and not the recorded that was found. the next version of this code would have included this optimization but as it stands now, it does not have this functionality.

Section IV (Code for finding a record):

```
Find() {
        //the variable called “record_file” is assigned the location of the file called “$localFile” with is our phantom database file (.data.txt)
  record_file=$localFile

        //this line of code prompts the user to input some letter or word associated with the record the user wishes to locate
  echo -n "Enter record to find: "

        //this line makes the shell script wait until the user enters something and be saved into “text”    variable
 read text

        //grep command  searches the record_file, ignores case (-i) and lists the line number where the search match was found (-n). The result of this search is saved into the "record" variable
  record="$(grep -in $text "$record_file")"

        //the if statement checks if the grep command successfully found any string entered by the user
   if [ $? -eq 0 ]; then

        //if the search was successful, displays the results
     echo "$record"

        //if the search was not successful then displays to the user “not found”
   else
     echo "Not found"
        //fi ends the if statement
   fi
}
```

Figure 4 shows a case where the user enters an entire record of a person to find it. The program displays the exact matching record when entered.
Figure 5 shows a case where the user enters some matching record that can be found in multiple matching records. In this case, the user enters the number “512” and it displays the multiple matches that have “512”
Figure 6 shows a case where when the user inputs a record and it doesn’t find anything. In this case, the user enters a name that is not in the database and it display a “Not found” message

Figure 4

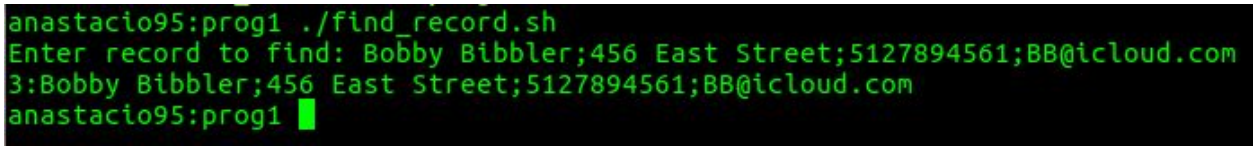


Figure 5

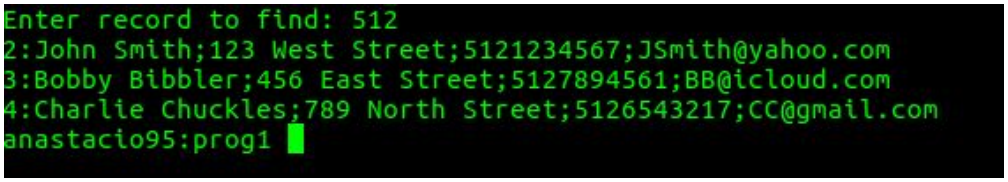


Figure 6