

## Homework# 4:

### Submission instructions:

1. Create a Google doc for your submission.
2. Start your responses from page 2 of the document and copy these instructions on page 1.
3. Fill in your name, campus ID and panther # in the fields provided. If this information is missing TWO POINTS WILL BE DEDUCTED.
4. Keep this page 1 intact. If this *submissions instructions* page is missing in your submission TWO POINTS WILL BE DEDUCTED.
5. Start your responses to each QUESTION on a new page.
6. If you are being asked to write code copy the code into a separate txt file and submit that as well. The code should be executable. E.g. if asked for a C program then provide myfile.c so that we can execute that script. In your answer to the specific question, provide the steps on how to execute your file (like a ReadMe).
7. If you are being asked to test code or run specific commands or scripts, provide the evidence of your outputs through a screenshot and/or screen video-recordings and copy the same into the document.
8. Upon completion, download a .PDF version of the google doc document and submit the same along with all the supplementary files (videos, pictures, scripts etc).
9. Scripts/Code without proper comments, indentation and titles (must have the name of the program, and name & email of the programmer on top the script).

Full Name: Cassandra Lundberg

Campus ID: clundberg3

Panther #: 002345582

**Question: 1** For question one I first chose the 10 commands I thought I used the most while programming in UNIX. Then I created a file named mandatabase.txt using vi text editor. I then generated each of the manuals of the commands I chose and redirected the output to the mandatabase.txt file. By using the double >>, I was able to append the redirected outputs to the end of the file, creating a copy of the manuals for each command one after another. I then added my name and the title of the program to the top of the file, saved, then exited.

Next I created a file named helpme.sh using the vi editor to output the manual page from the mandatabase.txt file of the corresponding command input. First I wrote the first line as the executable program pathname used to interpret the script as a bash shell, written as "#!/bin/bash". Then added my name and title of the program as a comment. I used the "echo" shell command to print the prompt for the user to enter a command to be searched for in the mandatabase.txt. Then the "read" command was utilized to read a line from input, (the command entered) and assign the input to a variable called entry. I then decided to also assign the mandatabase.txt file to a variable called database to provide easier utility later. Then I noticed how in the manual of each command, the page where the command starts, begins with the command's name in capital letters followed by a (1). For example if I were to type the command "man gawk" into the UNIX terminal, the entire page of gawk's utility pops up with the phrase "GAWK(1)" listed at the top. So knowing this information I reassigned the variable "entry" to be the input entry translated from lowercase characters to all uppercase characters using the "tr" command. To do this I pipelined the input entry's value to the input of the transformation from lowercase to uppercase characters. Now the value of the variable "entry" is the input entered changed to all uppercase characters.

Then I needed to check to see if the input entered exists in the mandatabase.txt file at all. I used the if-then-else-fi control statement to check to see if the "entry" variable appended with (1) in the mandatabase.txt file. So first I utilized the command "sed" along with the option "-n" to keep all alterations to an edit stream. The sed command took an input starting with the "entry" variable and appended (1) to that stream then printed the pattern space from mandatabase.txt (now a variable called "database"). Then the output of that sed command is pipelined to the input of a "more" command allowing the user to scroll through the list of files, a page at a time. The control statement took the value of this entire pipeline to check the existence of the command in the mandatabase.txt file.

The then statement, outputs the altered input stream edited in the sed command. Then the else statement outputs "sorry, i cannot help you" using the "echo" command. And lastly, the control statement ends with "fi".

To execute this program, I had to first change the permission settings using the "chmod" command to allow the user to read, write, and execute the program. Once this is done, the shell script can be executed using the "./" pathname in front of the shell file name to run the shell script file located in the current working directory.

Screenshot of the 10 unix commands' manuals copied onto the mandatabase.txt file:

```
clundberg3@gsuad.gsu.edu@snowball:~  
[clundberg3@gsuad.gsu.edu@snowball ~]$ vi mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man ls >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man pwd >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ vi mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man cd >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man cat >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man grep >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man awk >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man sed >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man cp >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man mv >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man chmod >> mandatabase.txt  
[clundberg3@gsuad.gsu.edu@snowball ~]$
```

Screenshot of mandatabase.txt file contents:

```
*****Mandatabase*****      Name of Programmer: Cassandra Lundberg  Email of Programmer: clundberg3@student.gsu.edu  
LS (1)                        User Commands                        LS (1)  
  
NAME  
    ls - list directory contents  
  
SYNOPSIS  
    ls [OPTION]... [FILE]...  
  
DESCRIPTION  
    List information about the FILES (the current directory by default).  Sort entries alphabetically if none of  
    -cftuvSUX nor --sort is specified.  
  
    Mandatory arguments to long options are mandatory for short options too.  
  
    -a, --all  
        do not ignore entries starting with .  
  
    -A, --almost-all  
        do not list implied . and ..  
  
    --author  
        with -l, print the author of each file  
  
    -b, --escape  
        print C-style escapes for nongraphic characters  
  
    --block-size=SIZE  
        scale sizes by SIZE before printing them; e.g., '--block-size=M' prints sizes in units of 1,048,576  
        bytes; see SIZE format below  
  
    -B, --ignore-backups  
        do not list implied entries ending with ~  
  
    -c      with -lt: sort by, and show, ctime (time of last modification of file status information); with -l:  
            show ctime and sort by name; otherwise: sort by ctime, newest first  
  
    -C      list entries by columns  
  
    --color[=WHEN]  
        colorize the output; WHEN can be 'never', 'auto', or 'always' (the default); more info below  
  
    -d, --directory  
        list directories themselves, not their contents  
  
2,0-1      Top
```

Screenshot of helpme.sh shell script:



```
clundberg3@gsuad.gsu.edu@snowball:~  
[clundberg3@gsuad.gsu.edu@snowball ~]$ ./helpme.sh  
Please type a command:  
rm  
sorry, I cannot help you.  
[clundberg3@gsuad.gsu.edu@snowball ~]$
```

**Question: 2** In question number 2 first I created a file named myexamfile.txt using vi editor, then added the title of the program and my name. Next I searched on google for a wikipedia page of one of my favorite games called "Minecraft". I highlighted the entire web page, pressed CTRL-C, then went to the text file and right clicked to paste the text to the file. I then saved and exited the file. Then I used the command "cd ~" to go to the home directory. I then created a new directory using the command "mkdir" and named it midterm by typing "mkdir midterm". Then to copy the file "myexamfile.txt" into the directory "midterm" is typed the command "cp myexamfile.txt midterm".

The next thing I did here was create a shell script file using vi editor called, "numberstatements.sh". Then I added my name and the title of the program as well as the first line as the executable program pathname used to interpret the script as a bash shell, written as "#!/bin/bash". Next I created a variable called "wiki" to stored the file "myexamfile.txt". Then I used the command "echo" to output an argument prompting for the number of sentences of the file myexamfile.txt. I decided it would be easier to store the count in a variable called "sentence" to later output the value of. To get the count, I utilized the command "grep" along with the options "-o" to find only the matching regex and the options "-c" to count the number of times the pattern matched. Then the variable "sentence" is outputted.

Next to find the number of words and characters of each regex "sentence" then I created a new variable called "actual\_sentence" to hold the value of the same grep statement but without the "-c" option to output the actual sentences that were searched for. Then utilizing the for-do-done control statement, for every x sentence in the list of sentences found in the variable "actual\_sentence", do find the word count of characters and the word count of letters. This is done by taking the value of x(each sentence) and pipelining the output to the input of the command "wc" where the option "-c" finds the count of characters and the option "-w" finds the count of words. Then the done part of the control statement ends the statement.

To execute the program the command "chmod" must be used to grant permissions to the user to read, write, and execute the program. Lastly to actually execute the shell script name must be prefixed with "." because the shell script file is located in the current working directory.

Screenshot of the shell script numberstatements.sh which finds the number of statements in a file and the number of words and characters for every statement.

```
clundberg3@gsuad.gsu.edu@snowball:~/midterm
#!/bin/bash
# Name of Programmer: Cassandra Lundberg Email of Programmer: clundberg3@student.gsu.edu
#Title: numberstatements.sh
# This shell script is designed to search for the number of statements in the file "myexamfile.txt" and display the number as well as
# find the number of words and letters of each statement in the file.

wiki="myexamfile.txt" #assigned the file to a variable named "wiki"

echo "The number of sentences separeated by a (.) is: " #prompt

sentence="$(grep -oc ^\..*\. $wiki)" #the variable sentence is the searched number of sentences in the file by finding any number of
#characters in between two periods
echo "$sentence" #outputs the number of sentences in the file
echo " "

actual_sentence="$(grep ^\..*\. $wiki)" #variable "actual_sentence" finds the all the statements where characters are in between two
#periods.

for x in $actual_sentence #for every statement in the list of statements, do
do
    echo "The number of letters are: "
    echo $x | wc -c #print the number of character for every statement

    echo "The number of words are: "
    echo $x | wc -w #print the number of words for every statement
done
```

Screenshot shows the new directory called “midterm” in the home directory. The midterm directory has the file called “myexamfile” which contains the wikipedia copied text and the “numberstatements.sh” shell script which contains the code to search the “myexamfile”.

```
clundberg3@gsuad.gsu.edu@snowball:~/midterm
[clundberg3@gsuad.gsu.edu@snowball ~]$ ls
addressbook.txt  Copies          foo.sh          helpme.sh       Lab4             mandatabase.txt  phonebook.sh    seddN3GEr       simple.sh
a.out           csc3320        gdrive         Homework_Files  midterm          myexamfile.txt  public          sedg5gRbg       test.txt
calculator.sh    directories.tar hello.c         index.html?.html=  myName.c        Result          sedk4vOpb       textfiles.tar
cfiles.tar      foo.class      hello.c        javafiles.tar    myName.c        Result          sedLEkYnr
checkError.sh   foo.java       hello.sh       Lab3
[clundberg3@gsuad.gsu.edu@snowball ~]$ cd midterm
[clundberg3@gsuad.gsu.edu@snowball midterm]$ ls
myexamfile.txt  numberstatements.sh
[clundberg3@gsuad.gsu.edu@snowball midterm]$
```

Screenshot shows the result of part a of the question where the number of statements in the file is output.

```
clundberg3@gsuad.gsu.edu@snowball:~/midterm
[clundberg3@gsuad.gsu.edu@snowball midterm]$ ./numberstatements.sh
The number of sentences separeated by a (.) is:
257
[clundberg3@gsuad.gsu.edu@snowball midterm]$
```

Screenshot shows the result of part b of the question which shows the number of words and letters in each statement.

```
clundberg3@gsuad.gsu.edu@snowball:~/midterm
3
The number of words are:
1
The number of letters are:
5
The number of words are:
1
The number of letters are:
9
The number of words are:
1
The number of letters are:
3
The number of words are:
1
The number of letters are:
9
The number of words are:
1
The number of letters are:
7
The number of words are:
1
The number of letters are:
4
The number of words are:
1
[clundberg3@gsuad.gsu.edu@snowball:~/midterm]$
```

Screenshot shows “myexamfile.txt”

```
clundberg3@gsuad.gsu.edu@snowball:~/midterm
minecraft
From Wikipedia, the free encyclopedia
Jump to navigationJump to search
This article is about the original video game. For the greater franchise, see Minecraft (franchise). For other uses, see Minecraft (disambiguation).
Minecraft
The default player skin, Steve, running across a grassy plain while carrying a Iron pickaxe. Alongside him is a tamed wolf. In the background, there is a pig, a chicken, a cow, a skeleton, a zombie, and a creeper. Mountains and cliffs fill the background, and the sky is blue, filled with clouds. Hovering over the scene is the Minecraft logo.
Promotional cover art
Developer(s) Mojang Studios[b]
Publisher(s)
Mojang Studios[c]
Xbox Game Studios[d]
Sony Interactive Entertainment[e]
Designer(s)
Markus Persson[f]
Jens Bergensten[g]
Artist(s)
Markus Toivonen
Jasper Boerstra
Composer(s) C418[h]
Series Minecraft
Platform(s)
Windows, macOS, Linux
Release
18 November 2011[a]
Genre(s) Sandbox, survival
Mode(s) Single-player, multiplayer
Minecraft is a sandbox video game developed by the Swedish video game developer Mojang Studios. The game was created by Markus "Notch" Persson in the Java programming language. Following several early private testing versions, it was first made public in May 2009 before fully releasing in November 2011, with Jens Bergensten then taking over development. Minecraft has since been ported to several other platforms and is the best-selling video game of all time, with over 200 million copies sold and over 140 million monthly active users as of 2021.[18]
In Minecraft, players explore a blocky, procedurally-generated 3D world with virtually infinite terrain, and may discover and extract raw materials, craft tools and items, and build structures or earthworks. Depending on game mode, players can fight computer-controlled mobs, as well as cooperate with or compete against other players in the same world. Game modes include a survival mode, in which players must acquire resources to build the world and maintain health, and a creative mode, where players have unlimited resources and access to flight. Players can modify the game to create new gameplay mechanics, items, and assets.
3
3
1,1 Top
```



**Question: 3** For question number 3, I first created a file named calculator.sh using vi editor. Then I wrote the first line as the executable program pathname used to interpret the script as a bash shell, written as “#!/bin/bash”. I entered as comments my name as the programmer and the title of the program, as well as the purpose. My first thought when designing this program was how to implement a prompt to enter the numbers and the operation desired. My solution was to utilize the if-then-elif-else-fi control statements. Then my attention was turned to how to implement the clear and cancel functions. So I thought about everytime I used a calculator and what exactly the clear and cancel functions did when I entered them. The cancel functions always took the user back one step, so if an entire operation was entered then going back one step would prompt for the number. Going back another step would prompt for an operation, and going back another step would prompt the beginning asking for the first number. The clear function on the other hand would always go back to the beginning. I decided the best approach to this was to create methods.

My program initially has the methods before the program actually starts computing. I output some fancy designs with asterisks and greeted the user. Then I called the method to begin the operation called “operation\_prompt()”. This method asked the user to enter an operation and informed the user of the conditions to exit the calculator and clear the calculator. The input in which the user enters is saved into the variable called “operation”. Using an if-then-elif-else-fi control statement, if the value of the input variable is equal to the string of characters “exit” then the exit command is executed which terminates the shell and exits the program. If the value of the input is equal to the string of characters “clear” where the user wants to clear the operation entered, then the operation\_prompt() method is called. This begins the prompt from the beginning and relays the options of the operations again. If the value of the input variable is equal to the string “cancel” then the prompt is cancelled and starts all over again so the operation\_prompt() method is called. In this method the cancel and clear “functions” of the calculator essentially do the same thing. When none of these inputs are entered then the else control statement allows the computation to continue by calling the number1\_prompt() method.

The number1\_prompt() method asks the user to enter the first number to be operated on by the operation already entered. The input is then read and saved into the variable called “number1” then output to the user to show what was entered. I used another if-then-elif-else-fi control statement to account for the cancel and clear functions of the calculator. If the input value entered is equal to the string “exit” then the shell terminates and the program is exited. Else if the input is equal to the string of characters “clear”, then the program starts from the beginning, and the operation\_prompt() is called. Else if the input is equal to the string “cancel” then the program goes back one step and the number1\_prompt() method is repeated to prompt a different value for the number1 variable. Otherwise if the exit, cancel, or clear function are not input, then the program continues to compute and calls the number2\_prompt() method.

The number2\_prompt() method performs the same logic as the number1\_prompt() method however there is no else control statement. When the number2\_prompt()

method ends then the rest of the code is executed after the fancy designs. The code ensures to continue executing as long as the input value for the operation variable is not equal to "exit". This is done utilizing the while-do-done control statement. While the operation variable is not equal to "exit" then do the calculations with the two numbers and one operator. If the input value entered for the operation is equal to "+" then the output of the two variables(number1 and number2) are pipelined into the input of the command "bc`" which stands for basic calculator that does basic mathematical calculations. So the command "bc" does a computation on the output of the two variables then assigns that output to a variable called add. The variable number 1, number2, and add are all output in the order of "number1 + number2 = add" as a calculator would show. The method operation\_prompt() is then called to repeat this process until the user decides to quit the program by typing in exit in any input of the program. This same logic is done for subtraction, division, modulo, and multiplication with the addition of the "\"operator in front of the multiplication sign "\*" to indicate multiplication and not regex. Else if none of the operators enters are equal to these, then the operation is invalid and tells the user "Operation Invalid" then the operation\_prompt() is called again. The command "fi" is to close the if -then-elif-else-fi control statement and the "done" command closes the while-do-done control statement. The program exits with the command "exit". I then changed the permissions using the command "chmod" to give the user to read, write, and execute the program. And lastly I executed using "./" before the shell script program name to account for the program being in the current working directory.

## Screenshots of code of calculator.sh

```
clundberg3@gsuad.gsu.edu@snowball:~  
$ ./bin/bash  
# Name of Programmer: Cassandra Lundberg Email of Programmer: clundberg3@student.gsu.edu  
# Title: calculator.sh  
# This shell script asks a user for two numbers and an operator and performs operations on the two numbers based on the specified  
# operation.  
  
operation_prompt(){ #operation prompt method to ask user which operation to use  
    echo " "  
    echo "Which operation would you like to perform?" #prompt for operation  
    echo "Please enter an operation:"  
    echo "Addition (+)"  
    echo "Subtraction (-)"  
    echo "Multiplication (*)"  
    echo "Division (/)"  
    echo "Modulo (%)"  
    echo "Enter exit anytime to exit!"  
    echo "Enter clear to clear entries | Enter cancel to reenter entry"  
    echo " "  
    read operation #reads the users input for an operation  
  
    if [ "$operation" = "exit" ] #if the user's input is exit then the program exists the calculator shell script  
    then  
        exit  
    elif [ "$operation" = "clear" ] #if the user's input is clear then the program returns to the beginning of the prompt  
    then  
        operation_prompt  
    elif [ "$operation" = "cancel" ] #if the user's input is cancel then the program returns to the beginning of the entry  
    then  
        operation_prompt  
    else #otherwise the prompt continues to the number1 prompt method  
        number1_prompt  
    fi  
}  
number1_prompt(){ #the number1 prompt gets the first number from the user's input  
    echo "Please enter the first number: "  
    echo " "  
    read number1 #get user's input  
    echo "You entered: " "$number1" #tells the user what they entered  
  
    if [ "$number1" = "exit" ] #if the value of number1 is exit then the program exists  
    then  
        exit  
    elif [ "$number1" = "clear" ] #if the value of number1 is clear then the program starts from the beginning of operation prompt  
    then  
        operation_prompt  
    fi  
}
```

1,1 Top

```

clundberg3@gsuad.gsu.edu@snowball:~
}
number1_prompt(){ #the number1 prompt gets the first number from the user's input
    echo "Please enter the first number: "
    echo " "
    read number1 #get user's input
    echo "You entered: " "$number1" #tells the user what they entered

    if [ "$number1" = "exit" ] #if the value of number1 is exit then the program exits
    then
        exit
    elif [ "$number1" = "clear" ] #if the value of number1 is clear then the program starts from the beginning of operation prompt
    then
        operation_prompt
    elif [ "$number1" = "cancel" ] #if the value of number1 is cancel then the method number1 prompt begins again
    then
        number1_prompt
    else #otherwise continue to method number2 prompt
        number2_prompt
    fi
}
number2_prompt(){ #the number2 prompt method gets the second number from the user's input
    echo "Please enter the second number: "
    echo " "
    read number2 #get the user's input
    echo "You entered: " "$number2" #repeat the user's entry

    if [ "$number2" = "exit" ] #if the value if number2 is exit then the program exits
    then
        exit
    elif [ "$number2" = "clear" ] #if the value of number2 is clear then the program starts from the beginning of operation prompt
    then
        operatin_prompt
    elif [ "$number2" = "cancel" ] #if the value of number2 is cancel then the method number2 prompt begins again
    then
        number2_prompt
    fi
}
echo " "
echo "Welcome to Cassie's Calculator!!" #beginning text for flare
echo "*****"
operation_prompt #calls on operation prompt

while [ "$operation" != "exit" ] #while the operation entered from user is not exit then perform the following
do

```

```
clundberg3@gsuad.gsu.edu@snowball:~
echo "Welcome to Cassie's Calculator!!" #beginning text for flare
echo "*****"
operation_prompt #calls on operation prompt

while [ "$operation" != "exit" ] #while the operation entered from user is not exit then perform the following
do
    if [ "$operation" = "+" ] #if the operation entered is "+"
    then
        add=`echo $number1 + $number2 | bc` #output the value of number1 + number2 to the input of the basic calculator command
        echo " "
        echo "$number1 + $number2 = $add" #print the expression and the result
        operation_prompt #jump to operation prompt method
    elif [ "$operation" = "-" ] #if the operation the user entered is "-"
    then
        sub=`echo $number1 - $number2 | bc` #output the value of number1 - number2 to the input of basic calculator command
        echo " "
        echo "$number1 - $number2 = $sub" #output the string expression and result in sub variable
        operation_prompt #jump to operation prompt method
    elif [ "$operation" = "*" ] #if the operation the user entered is "*"
    then
        mul=`echo $number1 \* $number2 | bc` #output the value of number1 * number2 to the input of basic calculator command
        echo " "
        echo "$number1 * $number2 = $mul" #output the string expression along with result of variable mul
        operation_prompt #return to operation prompt method
    elif [ "$operation" = "/" ] #if the operation the user entered is a "/"
    then
        div=`echo $((number1 / number2)) | bc` #the expression number1 / number2 must be a variable before outputting to basic
        #calculator command
        echo " "
        echo "$number1 / $number2 = $div" #output the string expression along with the result of variable div
        operation_prompt #return to operation prompt method
    elif [ "$operation" = "%" ] #if the operation entered by the user is "%"
    then
        mod=`echo $number1 % $number2 | bc` #output the value of number1 % number2 into the input of basic calculator command
        echo " "
        echo "$number1 % $number2 = $mod" #output the string expression and result variable mod
        operation_prompt #return to the operation prompt method
    else #otherwise the operation is invalid
        echo " "
        echo "Operation Invalid"
        operation_prompt #return to the operation prompt method
    fi
done #close while loop
exit #exit program
```

71,1 94%

Screenshots of operations:

## Addition:

```
clundberg3@gsuad.gsu.edu@snowball:~  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entries | Enter cancel to reenter entry  
  
exit  
[clundberg3@gsuad.gsu.edu@snowball ~]$ vi calculator.sh  
[clundberg3@gsuad.gsu.edu@snowball ~]$ ./calculator.sh  
  
Welcome to Cassie's Calculator!!  
*****  
  
Which operation would you like to perform?  
Please enter an operation:  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entries | Enter cancel to reenter entry  
  
+  
Please enter the first number:  
  
1  
You entered: 1  
Please enter the second number:  
  
4  
You entered: 4  
  
1 + 4 = 5  
  
Which operation would you like to perform?  
Please enter an operation:  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entries | Enter cancel to reenter entry
```

## Subtraction:

```
clundberg3@gsuad.gsu.edu@snowball:~  
+  
Please enter the first number:  
1  
You entered: 1  
Please enter the second number:  
4  
You entered: 4  
1 + 4 = 5  
  
Which operation would you like to perform?  
Please enter an operation:  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entrys | Enter cancel to reenter entry  
-  
Please enter the first number:  
8  
You entered: 8  
Please enter the second number:  
5  
You entered: 5  
8 - 5 = 3  
  
Which operation would you like to perform?  
Please enter an operation:  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entrys | Enter cancel to reenter entry
```

## Multiplication:

```
clundberg3@gsuad.gsu.edu@snowball:~
-
Please enter the first number:
8
You entered: 8
Please enter the second number:
5
You entered: 5
8 - 5 = 3

Which operation would you like to perform?
Please enter an operation:
Addition (+)
Subtraction (-)
Multiplication (*)
Division (/)
Modulo (%)
Enter exit anytime to exit!
Enter clear to clear entrys | Enter cancel to reenter entry

*
Please enter the first number:
4
You entered: 4
Please enter the second number:
6
You entered: 6
4 * 6 = 24

Which operation would you like to perform?
Please enter an operation:
Addition (+)
Subtraction (-)
Multiplication (*)
Division (/)
Modulo (%)
Enter exit anytime to exit!
Enter clear to clear entrys | Enter cancel to reenter entry
```



## Division:

```
clundberg3@gsuad.gsu.edu@snowball:~  
*  
Please enter the first number:  
4  
You entered: 4  
Please enter the second number:  
6  
You entered: 6  
4 * 6 = 24  
  
Which operation would you like to perform?  
Please enter an operation:  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entrys | Enter cancel to reenter entry  
  
/  
Please enter the first number:  
12  
You entered: 12  
Please enter the second number:  
4  
You entered: 4  
12 / 4 = 3  
  
Which operation would you like to perform?  
Please enter an operation:  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entrys | Enter cancel to reenter entry
```

## Modulo:

```
clundberg3@gsuad.gsu.edu@snowball:~  
/  
Please enter the first number:  
12  
You entered: 12  
Please enter the second number:  
4  
You entered: 4  
12 / 4 = 3  
  
Which operation would you like to perform?  
Please enter an operation:  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entrys | Enter cancel to reenter entry  
%  
Please enter the first number:  
8  
You entered: 8  
Please enter the second number:  
6  
You entered: 6  
8 % 6 = 2  
  
Which operation would you like to perform?  
Please enter an operation:  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entrys | Enter cancel to reenter entry
```

## Clear, Cancel, and Exit Functions:

```
clundberg3@gsuad.gsu.edu@snowball:~  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entrys | Enter cancel to reenter entry  
  
+  
Please enter the first number:  
  
clear  
You entered:  clear  
  
Which operation would you like to perform?  
Please enter an operation:  
Addition (+)  
Subtraction (-)  
Multiplication (*)  
Division (/)  
Modulo (%)  
Enter exit anytime to exit!  
Enter clear to clear entrys | Enter cancel to reenter entry  
  
+  
Please enter the first number:  
  
cancel  
You entered:  cancel  
Please enter the first number:  
  
exit  
You entered:  exit  
[clundberg3@gsuad.gsu.edu@snowball ~]$
```

**Question: 4** In question four, I imagined the best way to work on this type of problem was to create a text file of a database as in question 1 then create a shell script program which utilized that text file database of the address book. So first I used vi editor to create the database text file called addressbook.txt. Then I added the title of the file, my name, and my email, saved the file and exited. I then decided to begin working on the phonebook and created a shell script called phonebook.sh using vi editor. I added my name and title of the program at the top along with the first line as the executable program pathname used to interpret the script as a bash shell, written as `#!/bin/bash`.

I then decided to create a similar prompt to the calculator.sh shell script but instead implement a core `main_menu()` method. This `main_menu()` method included my special design of the address book to add flare. After the design I put the list of options to choose from in the method, numbered to create ease. I listed 1. Display 2. Search 3.Delete 4.Modify 5.Add and 6.Leave to exit the program. Then I used the `read` command to read an input and save the value as a variable called `choice`.

The program then called the `main_menu()` method. A while-do-done control statement is implemented to account for if the user decides to exit the program then the number 6 would remove the execution of the program from the while loop and exit. So the while statement concludes, while the value of the variable `choice` is not equal to 6 then do everything in the while loop. I also created a variable called `Book` to hold the text file addressbook.txt to utilize the file better. After looking back on the calculator program for ideas, I decided to not use an if-then-elif-else-fi control statement, since I would most likely be using if control statements within each individual computation of an option and multiple if control statements would look messy and confusing. So I decided the best decision is to perform a case-in-esac control statement. This way I can utilize the value in the variable `choice` to support a multiway branching.

In choice number 1) `Display`, I decided to portray the address book with each tuple entry as `firstname; lastname; phonenumber; address`. By doing this each line is an entry of a first name, last name, phone number, and an address. Each attribute or field is separated by a semi-colon. Since the display of the addressbook.txt has to have each entry in alphabetical order, I decided to place them in order by first name, which is the first field. I utilized the `sort` command to divide each field by a semi-colon, sort by the first field which is the first name, and ignore the case of each entry on the variable `Book` which stores the text file addressbook.txt. This output is then pipelined to the input of the command `cat` which outputs the sorted addressbook.txt file with numbered lines to represent entries. Then the `main_menu()` method is called to allow the user to choose another option.

Then in choice number 2), the user is prompted to enter the first name or phone number of the contact to be searched for. This input is read and the value is saved to the variable `find`. I then used the command `grep` along with the option `-i` to ignore cases of the variable. The command searches the entire text file addressbook.txt for the contact with the first name or the phone number entered. Then the `main_menu()`

method is called to allow the user to choose another option.

The third choice 3), to delete a contact required much more thought. I prompted the user to enter the first name of a contact in which they want to delete and read the input to a variable called "fname". Then I used the "grep" command to search for the line with the contact in the addressbook.txt along with an if-then-else-fi control statement. If the line with the name searched for from the input variable "fname" exists then the user is prompted to confirm the deletion of that contact. The input confirmation is stored in a variable called "confirmation". Another if-then-else-fi control statement is used nested in the one before to check if the value of the input variable is equal to "y". If so then the command "sed" along with the option "-i" to edit the file in place, is used to delete the line with the input variable "fname" and not output to the command line but edits the actual file. Then the output "The contact has been deleted" is printed to ensure the user of the confirmation and the main\_menu() method is called. Otherwise the "confirmation" variable is "n" and the user is ensured that "the contact has not been deleted". The main\_menu() method is then called and that control statement ends. Else if the command "grep" does not find the value with the contact to be deleted then the user is told "There are no such contacts that match the entered name or phone number" and the main\_menu() method is called.

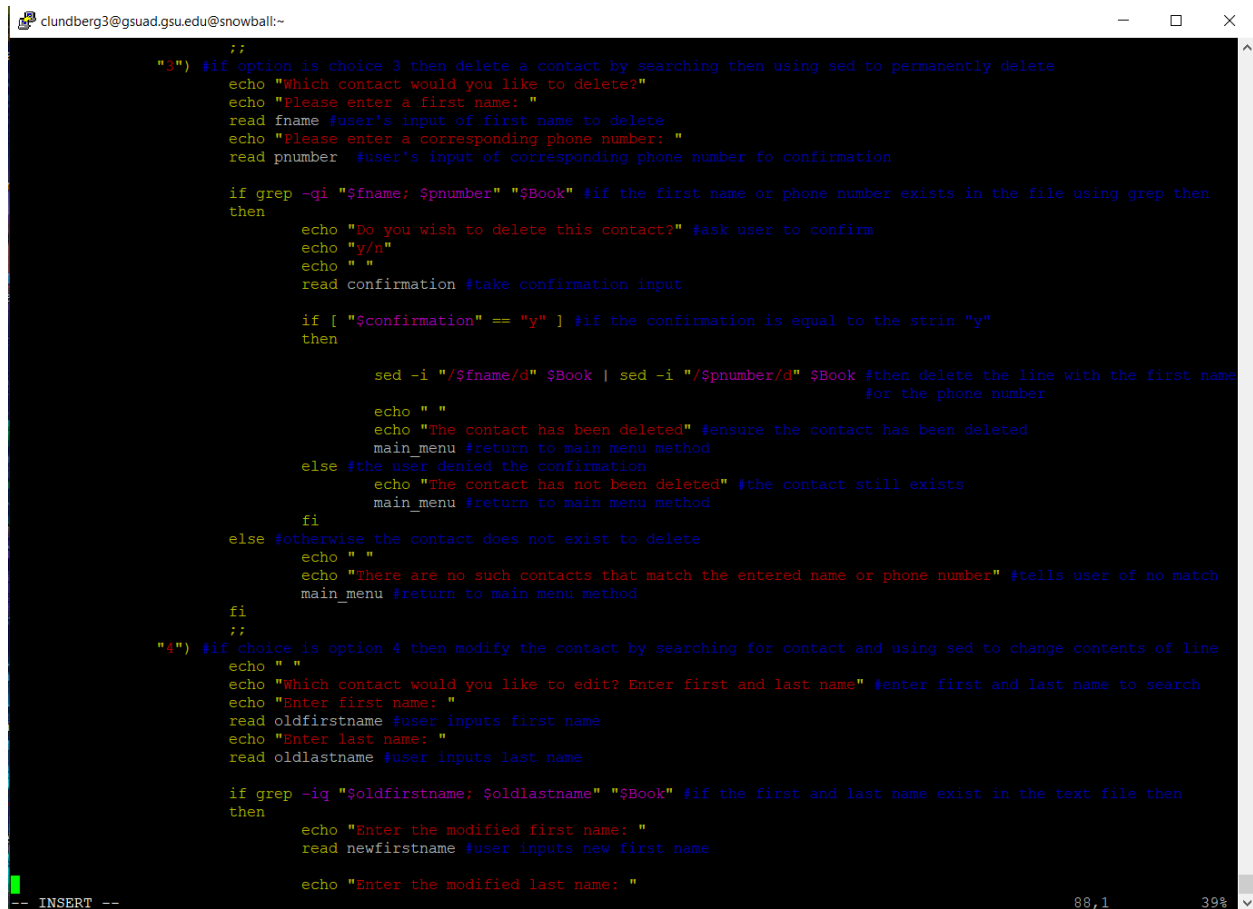
With choice 4), I thought about allowing the user to alter one field of a line but then discovered the problem of if the user wants to modify multiple fields of the contact's information. So I figured, if the user wants to change a contact, they must enter all new fields into the contact. This is similar to deleting a contact then adding another new contact. The user is prompted to enter a first and last name then the values are stored in variables "firstname" and "lastname". Then similar to the search choice, grep is used to search for both values of variables in the addressbook.txt file. An if-then-else-fi control statement is used. The program then prompts the user to enter in new fields for each of the four fields of the contact. Each field is saved as a new individual variable. The user is then prompted to confirm the new field alterations of the contact. Once the confirmation input is read and saved in a variable then another if-then-else-fi control statement is utilized. If the conformation variable is equal to "y" then the command "sed" is used with the "-i" edit file in place option to change the line with the input value of the oldfirstname with the newfirstname; newlastname; newphonenum; newaddress, each separated by semi-colons. Then the user is confirmed of the change with an output statement and the main\_menu() method is called. Otherwise the user did not confirm the modification, so the user is informed of no change through an output statement as well as the main\_menu() method being called. Else if the program cannot find the line with the oldfirstname entered into the input of the variable, then the user is informed that the contact does not exist and the main\_menu() method is called.

When working on choice 5) I decided the best way to add a new entry to the addressbook.txt was to input each field and save each to a new variable to later be redirected into the file. So the user is asked to enter the firstname and it is saved as the variable "firstname", and enter the lastname saved as "lastname", and the phone number and address. Then the user is asked to confirm the entry as the fields are

output to display the new contact. The confirmation is read in another variable and an if-then-else-fi control statement is utilized to check the confirmation. If the confirmation variable is equal to “y” then the output of firstname; lastname; phonenumber; address are all redirected to the variable storing the text file addressbook.txt . The semi-colons are also redirected into the file and the user is ensured of the addition of the addressbook.txt file along with the calling of the main\_menu() method afterwards. Otherwise if the user does not wish to add the new entry to the addressbook.txt then the user is ensured of no addition and the main\_menu() method is called.

The esac part of the control statement case-in-esac ensures the ending of the control statement and the done ensures the end of the while-do-done control statement. The entire program is looped until the number 6 is entered into the choice of the main\_menu() method. Lastly to ensure the permissions for the addressbook.txt database is only accessible to the user where the user can read, write, and execute the program the command “chmod 700 .addressbook.txt” is typed. To execute the phonebook.sh shell script the permissions must be changed using the same permissions command then the script must be prefixed with “.” because it is located in the current working directory.

## Screenshots of phonebook.sh



```
clundberg3@gsuad.gsu.edu@snowball:~  
;;  
"3") #if option is choice 3 then delete a contact by searching then using sed to permanently delete  
echo "Which contact would you like to delete?"  
echo "Please enter a first name: "  
read fname #user's input of first name to delete  
echo "Please enter a corresponding phone number: "  
read pnumber #user's input of corresponding phone number fo confirmation  
  
if grep -qi "$fname; $pnumber" "$Book" #if the first name or phone number exists in the file using grep then  
then  
    echo "Do you wish to delete this contact?" #ask user to confirm  
    echo "y/n"  
    echo " "  
    read confirmation #take confirmation input  
  
    if [ "$confirmation" == "y" ] #if the confirmation is equal to the strin "y"  
    then  
        sed -i "/$fname/d" $Book | sed -i "/$pnumber/d" $Book #then delete the line with the first name  
                                                #for the phone number  
        echo " "  
        echo "The contact has been deleted" #ensure the contact has been deleted  
        main_menu #return to main menu method  
    else #the user denied the confirmation  
        echo "The contact has not been deleted" #the contact still exists  
        main_menu #return to main menu method  
    fi  
else #otherwise the contact does not exist to delete  
    echo " "  
    echo "There are no such contacts that match the entered name or phone number" #tells user of no match  
    main_menu #return to main menu method  
fi  
;;  
"4") #if choice is option 4 then modify the contact by searching for contact and using sed to change contents of line  
echo " "  
echo "Which contact would you like to edit? Enter first and last name" #enter first and last name to search  
echo "Enter first name: "  
read oldfirstname #user inputs first name  
echo "Enter last name: "  
read oldlastname #user inputs last name  
  
if grep -iq "$oldfirstname; $oldlastname" "$Book" #if the first and last name exist in the text file then  
then  
    echo "Enter the modified first name: "  
    read newfirstname #user inputs new first name  
  
    echo "Enter the modified last name: "  
    read newlastname #user inputs new last name  
  
    sed -i "/$oldfirstname; $oldlastname/ s/$oldfirstname $oldlastname/$newfirstname $newlastname/" $Book  
    echo "Contact modified successfully"  
    main_menu #return to main menu method  
fi  
fi  
;;  
"6") #if choice is option 6 then exit the script  
echo "Exiting script..."  
exit 0  
fi  
done
```

```

;;
"4") #if choice is option 4 then modify the contact by searching for contact and using sed to change contents of line
echo " "
echo "Which contact would you like to edit? Enter first and last name" #enter first and last name to search
echo "Enter first name: "
read oldfirstname #user inputs first name
echo "Enter last name: "
read oldlastname #user inputs last name

if grep -iq "$oldfirstname; $oldlastname" "$Book" #if the first and last name exist in the text file then
then
    echo "Enter the modified first name: "
    read newfirstname #user inputs new first name

    echo "Enter the modified last name: "
    read newlastname #user inputs new last name

    echo "Enter the modified phone number: "
    read newphonenumber #user inputs new phone number

    echo "Enter the modified address: "
    read newaddress #user inputs new address

    echo "Change the contact to: " #ask user to confirm new fields of contact
    echo -e "$newfirstname; $newlastname; $newphonenumber; $newaddress" #print the fields of contact
    echo "y/n"
    echo " "
    read confirmation #input user's confirmation

    if [ "$confirmation" == "y" ] #if user's confirmation is "y" then
    then
        sed -i "$oldfirstname/c$newfirstname; $newlastname; $newphonenumber; $newaddress" "$Book"
        #then change the line with the original first name to the new fields of the contact
        echo "The contact information was changed" #tell the user that the fields have changed
        main_menu #return to main menu method
    else #otherwise the user does not confirm modification
        echo "The contact information was not changed" #inform user of no change
        main_menu #return to main menu method
    fi

else #otherwise the contact does not exist to modify
    echo "The contact does not exist to modify"
    main_menu #return to main menu method
fi
;;

"5") #if the choice is option 5 then a new contact is added by creating new fields of a line and redirecting them to
#the text file

```

```
clundberg3@gsuad.gsu.edu@snowball:~  
  
    then  
        sed -i "/$oldfirstname/c$newfirstname; $newlastname; $newphonenumber; $newaddress" "$Book"  
        #then change the line with the original first name to the new fields of the contact  
        echo "The contact information was changed" #tell the user that the fields have changed  
        main_menu #return to main menu method  
    else #otherwise the user does not confirm modification  
        echo "The contact information was not changed" #inform user of no change  
        main_menu #return to main menu method  
    fi  
  
    else #otherwise the contact does not exist to modify  
        echo "The contact does not exist to modify"  
        main_menu #return to main menu method  
    fi  
;;  
"5") #if the choice is option 5 then a new contact is added by creating new fields of a line an redirecting them to  
#the text file  
    echo "Enter the first name: "  
    read firstname #enter the first name of the new contact  
  
    echo "Enter the last name: "  
    read lastname #enter the last name of the new contact  
  
    echo "Enter the phone number: "  
    read phonenumber #enter the new phone number of the contact  
  
    echo "Enter the address: "  
    read address #enter the address of the new contact  
  
    echo " "  
    echo "Do you wish to enter the values? " #ask user to confirm new contact information  
    echo -e "$firstname; $lastname; $phonenumber; $address" #print information of contact  
    echo "y/n"  
    echo " "  
    read confirmation #take user's input of confirmation  
  
    if [ "$confirmation" == "y" ] #if the confirmation is equal to the string "y" then  
    then  
        echo "$firstname; $lastname; $phonenumber; $address" >>$Book #redirect all the fields to the text file  
        echo "The values were written to the address book" #tell the user of the entry  
    else #otherwise the entry is not entered into address book  
        echo "The values were not entered" #tell the user the entry was cancelled  
    fi  
    main_menu #return to the main menu method  
;;  
esac  
done
```

Screenshots of the options:



## Displaying and Searching:

```
clundberg3@gsuad.gsu.edu@snowball:~  
6  
[clundberg3@gsuad.gsu.edu@snowball ~]$ vi phonebook.sh  
[clundberg3@gsuad.gsu.edu@snowball ~]$ ./phonebook.sh  
*****  
Cassie's Phone Book  
*****  
  
Please enter a number:  
1.Display  
2.Search  
3.Delete  
4.Modify  
5.Add  
6.Leave  
1  
  
1  ***Address Book***  
2  Cassie; Lundberg; 770-652-0720; 1959 Amber Trail Duluth, GA 30096  
3  Josephine; Lundberg; 123-456-7890; 1265 Bramlett Blvd Lawrenceville, GA 30045  
4  John; Bray; 678-464-4935; 1959 Amber Trail Duluth, GA 30096  
  
*****  
Cassie's Phone Book  
*****  
  
Please enter a number:  
1.Display  
2.Search  
3.Delete  
4.Modify  
5.Add  
6.Leave  
2  
What is the first name or phone number of the contact to search?  
John  
John; Bray; 678-464-4935; 1959 Amber Trail Duluth, GA 30096  
*****  
Cassie's Phone Book  
*****  
  
Please enter a number:  
1.Display  
2.Search  
3.Delete  
4.Modify  
5.Add  
6.Leave  
3
```

Deleting: Screenshot shows the display before delete, the confirmation of delete and the after display of the delete option.

```
clundberg3@gsuad.gsu.edu@snowball:~  
1  ***Address Book***  
2  Cassie; Lundberg; 770-652-0720; 1959 Amber Trail Duluth, GA 30096  
3  Josephine; Lundberg; 123-456-7890; 1265 Bramlett Blvd Lawrenceville, GA 30045  
4  John; Bray; 678-464-4935; 1959 Amber Trail Duluth, GA 30096  
  
*****  
Cassie's Phone Book  
*****  
  
Please enter a number:  
1.Display  
2.Search  
3.Delete  
4.Modify  
5.Add  
6.Leave  
3  
Which contact would you like to delete?  
Please enter a first name:  
Cassie  
Do you wish to delete this contact?  
y/n  
y  
  
The contact has been deleted  
*****  
Cassie's Phone Book  
*****  
  
Please enter a number:  
1.Display  
2.Search  
3.Delete  
4.Modify  
5.Add  
6.Leave  
1  
  
1  ***Address Book***  
2  Josephine; Lundberg; 123-456-7890; 1265 Bramlett Blvd Lawrenceville, GA 30045  
3  John; Bray; 678-464-4935; 1959 Amber Trail Duluth, GA 30096  
  
*****  
Cassie's Phone Book  
*****
```

Adding: Screenshot shows adding the contact back into the addressbook.txt after deleting it and the display of the contact back in the text file.

```
clundberg3@gsuad.gsu.edu@snowball:~  
3 John; Bray; 678-464-4935; 1959 Amber Trail Duluth, GA 30096  
*****  
Cassie's Phone Book  
*****  
Please enter a number:  
1.Display  
2.Search  
3.Delete  
4.Modify  
5.Add  
6.Leave  
5  
Enter the first name:  
Cassandra  
Enter the last name:  
Lundberg  
Enter the phone number:  
770-652-0720  
Enter the address:  
1959 amber Trail Duluth, GA 30096  
Do you wish to enter the values?  
Cassandra; Lundberg; 770-652-0720; 1959 amber Trail Duluth, GA 30096  
y/n  
y  
The values were written to the address book  
*****  
Cassie's Phone Book  
*****  
Please enter a number:  
1.Display  
2.Search  
3.Delete  
4.Modify  
5.Add  
6.Leave  
1  
1 ***Address Book***  
2 Josephine; Lundberg; 123-456-7890; 1265 Bramlett Blvd Lawrenceville, GA 30045  
3 John; Bray; 678-464-4935; 1959 Amber Trail Duluth, GA 30096  
4 Cassandra; Lundberg; 770-652-0720; 1959 amber Trail Duluth, GA 30096  
*****
```

Modifying: Screenshot shows modifications to the Cassandra contact in changing the first name to Cassie, changing the last name to lundberg, and the phone number to 770-652-0270 and the address to all capital letters.

```
clundberg3@gsuad.gsu.edu@snowball:~
*****
Please enter a number:
1.Display
2.Search
3.Delete
4.Modify
5.Add
6.Leave
4

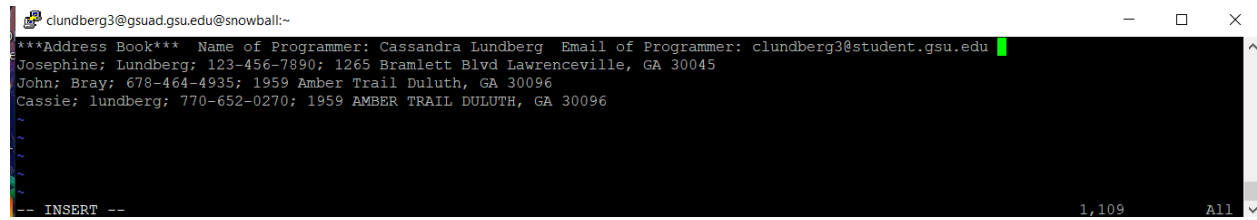
Which contact would you like to edit? Enter first and last name
Enter first name:
Cassandra
Enter last name:
Lundberg
Enter the modified first name:
Cassie
Enter the modified last name:
lundberg
Enter the modified phone number:
770-652-0270
Enter the modified address:
1959 AMBER TRAIL DULUTH, GA 30096
Change the contact to:
Cassie; lundberg; 770-652-0270; 1959 AMBER TRAIL DULUTH, GA 30096
y/n
y
The contact information was changed
*****
Cassie's Phone Book
*****
Please enter a number:
1.Display
2.Search
3.Delete
4.Modify
5.Add
6.Leave
1

1  ***Address Book***
2  Josephine; Lundberg; 123-456-7890; 1265 Bramlett Blvd Lawrenceville, GA 30045
3  John; Bray; 678-464-4935; 1959 Amber Trail Duluth, GA 30096
4  Cassie; lundberg; 770-652-0270; 1959 AMBER TRAIL DULUTH, GA 30096
```

Leave: Screenshot shows what happens when the leave option (6) is entered:

```
clundberg3@gsuad.gsu.edu@snowball:~
*****
Cassie's Phone Book
*****
Please enter a number:
1.Display
2.Search
3.Delete
4.Modify
5.Add
6.Leave
6
[clundberg3@gsuad.gsu.edu@snowball ~]$
```

Screenshot shows addressbook.txt:



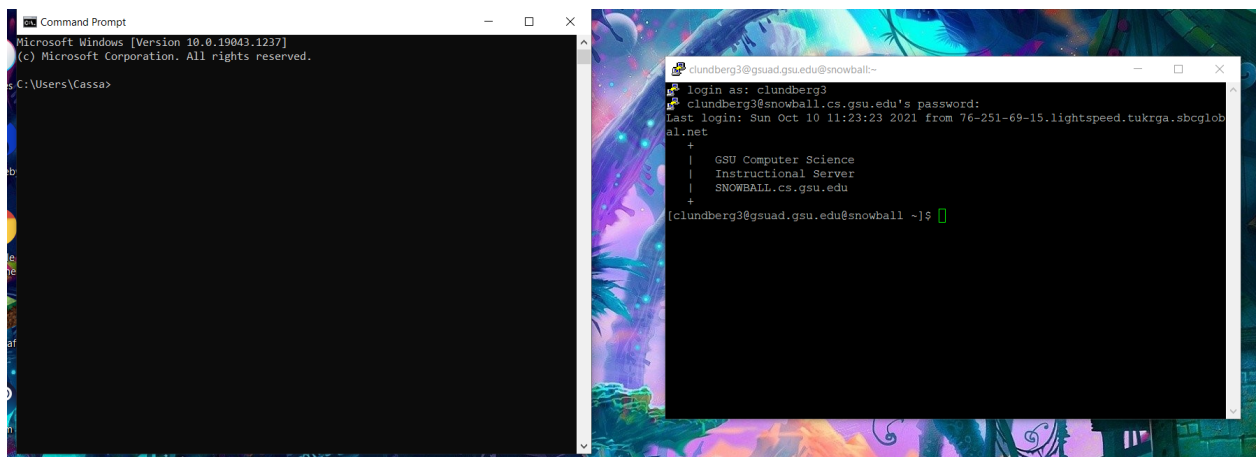
A screenshot of a terminal window with a black background and white text. The window title bar shows the user 'clundberg3@gsuad.gsu.edu@snowball:~'. The terminal content displays the output of a program, starting with a separator line '\*\*\*Address Book\*\*\*'. It then shows the name and email of the programmer, followed by three entries: Josephine Lundberg, John Bray, and Cassie Lundberg, each with their phone number and address. The bottom status bar of the terminal shows '-- INSERT --' on the left, '1,109' in the center, and 'All' on the right.

```
clundberg3@gsuad.gsu.edu@snowball:~
***Address Book***  Name of Programmer: Cassandra Lundberg  Email of Programmer: clundberg3@student.gsu.edu
Josephine; Lundberg; 123-456-7890; 1265 Bramlett Blvd Lawrenceville, GA 30045
John; Bray; 678-464-4935; 1959 Amber Trail Duluth, GA 30096
Cassie; lundberg; 770-652-0270; 1959 AMBER TRAIL DULUTH, GA 30096
-- INSERT -- 1,109 All
```

### Question: 5

- A. Shell is an interface which takes commands input from a keyboard and hands them to the operating system. Since a shell acts as an interpreter between commands from the user and the operating system, it is extremely important just to simply use a computer. Some examples of utilities a shell offers are; file management, programming, writing, editing, and saving text files, and assigning permissions to files.

B.



One of the biggest differences between the PC terminal and the snowball server terminal is the starter icon. The PC terminal shows C:\Users\myuser name> then the cursor is blinking. However on the snowball server, one I sign in, the cursor is a “\$” to represent that the shell is ready to read a command. This is because Windows uses a PowerShell and the snowball server uses a Bash shell.

- C. Since C is a compiled language it requires a compiler to translate the source code from C programming language to a machine language code. C also needs a text editor or code editor for writing and saving C code. For hardware it requires at least CPU and RAM memory as well as input/ output devices such as a keyboard, monitor, and mouse for entering commands.
- D. As “echo” and “printf( )” are both built in commands, they are different in the way where echo always exits with a zero status while printf( ) can give a non zero exit code status. Printf( ) also gives more control over the output format and allows for definition of a formatting string. Visually “echo” has a default new line character

but “printf()” requires it to manually be added.

- E. The “ssh” command is a program for logging into a remote machine and is intended to provide secure encrypted communications between two untrusted hosts over an insecure network.

```
clundberg3@gsuad.gsu.edu@snowball:~  
[clundberg3@gsuad.gsu.edu@snowball ~]$ ssh  
usage: ssh [-1246AaCfGgKkMnNqsTtVvXxYy] [-b bind_address] [-c cipher_spec]  
        [-D [bind_address:]port] [-E log_file] [-e escape_char]  
        [-F configfile] [-I pkcs11] [-i identity_file]  
        [-J [user@]host[:port]] [-L address] [-l login_name] [-m mac_spec]  
        [-O ctl_cmd] [-o option] [-p port] [-Q query_option] [-R address]  
        [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]  
        [user@]hostname [command]  
[clundberg3@gsuad.gsu.edu@snowball ~]$
```

The “scp” command copies files between hosts on a network and is a tool used by “ssh” network protocol.

```
clundberg3@gsuad.gsu.edu@snowball:~  
[clundberg3@gsuad.gsu.edu@snowball ~]$ scp  
usage: scp [-12346BCpqrv] [-c cipher] [-F ssh_config] [-i identity_file]  
        [-l limit] [-o ssh_option] [-P port] [-S program]  
        [[user@]host1:]file1 ... [[user@]host2:]file2  
[clundberg3@gsuad.gsu.edu@snowball ~]$ man scp  
[clundberg3@gsuad.gsu.edu@snowball ~]$  
[clundberg3@gsuad.gsu.edu@snowball ~]$ scp  
usage: scp [-12346BCpqrv] [-c cipher] [-F ssh_config] [-i identity_file]  
        [-l limit] [-o ssh_option] [-P port] [-S program]  
        [[user@]host1:]file1 ... [[user@]host2:]file2  
[clundberg3@gsuad.gsu.edu@snowball ~]$
```

The “wget” command is a utility for non-interactive download of files from the web by using the pasted URL from the server.

```
clundberg3@gsuad.gsu.edu@snowball:~  
[clundberg3@gsuad.gsu.edu@snowball ~]$ wget https://classic.minecraft.net/?html=&join=G2BRwuxVKM4LKwvZ  
[1] 26600  
[clundberg3@gsuad.gsu.edu@snowball ~]$ --2021-10-10 21:35:19-- https://classic.minecraft.net/?html=  
Resolving classic.minecraft.net (classic.minecraft.net)... 13.225.194.17, 13.225.194.127, 13.225.194.49  
, ...  
Connecting to classic.minecraft.net (classic.minecraft.net)|13.225.194.17|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 1739 (1.7K) [text/html]  
Saving to: 'index.html?.html='  
  
100%[=====>] 1,739      --.-K/s   in 0s  
  
2021-10-10 21:35:19 (223 MB/s) - 'index.html?.html=' saved [1739/1739]  
  
[clundberg3@gsuad.gsu.edu@snowball ~]$
```

P. S. This is super cool! I just downloaded a HTML version of the game Minecraft in one of its oldest releases onto the snowball server!!