CSE333 OPERATING SYSTEMS PROJECT 1 REPORT

Ekin NOHUTÇU - 150116067
Merve AYER - 150119828
Zeynep Naz AKYOKUŞ -150119073

MENU

```
control=1

while [ $control != 0 ]
do
    echo " MAIN MENU "
    echo "-----------------------------------"

    echo "Please select an option : "
    echo
    echo " 1. Create Histogram "
    echo " 2. Encryption"
    echo " 3. Delete oldest "
    echo " 4. Convert numbers"
    echo " 5. Organized files "
    echo " 6. Exit "
    echo -n "Enter your choice [1-6] : "

    read choice

    clear

    if [ $choice -eq 1 ]
    then

        ./myprog1.sh

    elif [ $choice -eq 2 ]
    then
        read -p " Enter the string and encryption key : " string key
        ./myprog2.sh $string $key

    elif [ $choice -eq 3 ]
    then
        ./myprog3.sh


    elif [ $choice -eq 4 ]
    then
        ./myprog4.sh

    elif [ $choice -eq 5 ]
    then
        ./myprog5.sh

    elif [ $choice -eq 6 ]
    then
        exit 1


    control=0
    fi
done
```

```
merve@merveayer:~$ cd Desktop
merve@merveayer:~/Desktop$ cd opsisproje
merve@merveayer:~/Desktop/opsisproje$ chmod +x menu.sh
merve@merveayer:~/Desktop/opsisproje$ ./menu.sh
 MAIN MENU
-----------------------------------
Please select an option :

 1. Create Histogram
 2. Encryption
 3. Delete oldest
 4. Convert numbers
 5. Organized files
 6. Exit
Enter your choice [1-6] :
```

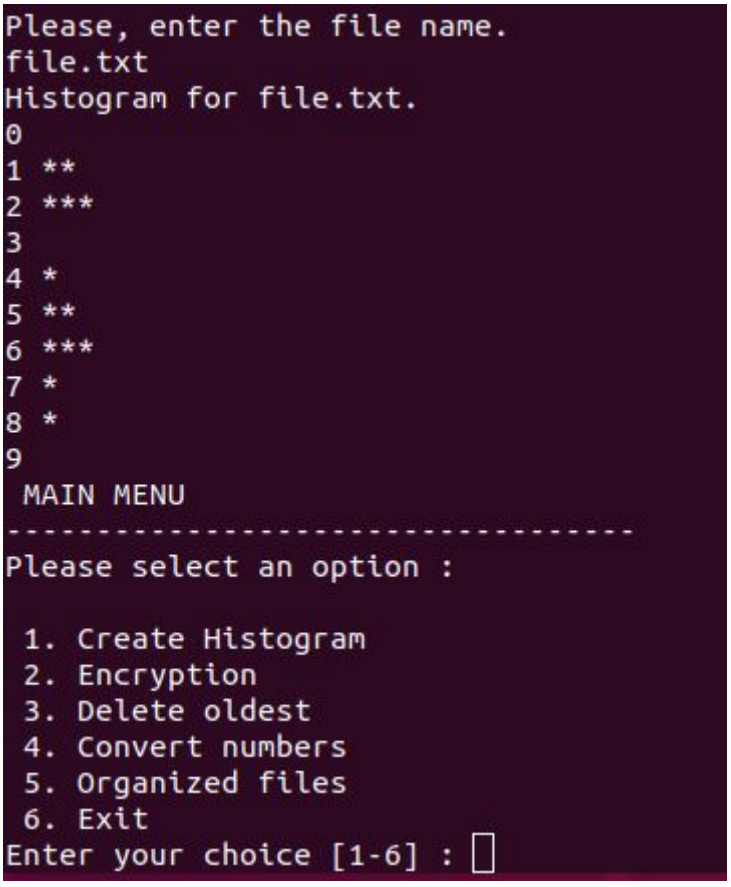
To execute all programs, we have designed a menu part. It should print the same menu after each program execution finished. To do that we used the while loop with a **control** variable. If the control variable is not 0, it will run the menu again and again.

We have 6 choices in the menu part. To choose the cases (programs), we defined a choice variable. **read choice** part will take the input from the user and put it into the choice variable. Depending on choice value, we used if else to execute corresponding programs. Each part of if else runs the corresponding .sh files.
To exit the menu part, you should enter the 6.

1- Create Histogram

```
echo "Please, enter the file name."
read file                                                                       # get the file name.



if [[ -n "$file" && `cat "$file"` ]];                                           # check if the file exist
then
        invalid=""                                                              # to show invalid numbers end of the program
        countArray=('0 ' '1 ' '2 ' '3 ' '4 ' '5 ' '6 ' '7 ' '8 ' '9 ')         # array for saving numbers and count of numbers as "(count)X(*)".

        echo "Histogram for $file."



        while read -r number; do                                                # read the file line by line and save the value to number
                if [[ "$number" -gt 10 || "$number" -lt 0 ]];                   # check if the number is in range 0-9, if is save it to invalid variable. Even if there are invalid values,
the program continues
                                                                                        #to run without considering them.
                then
                        invalid="$invalid $number"
                else                                                            # if number in range, add an "*" to countArray.
                        countArray[$number]+="*"
                fi
        done < $file


        for i in "${countArray[@]}"; do echo "$i"; done                         # to print the histogram.



        if [[ -n "$invalid" ]];                                                 # to show if there are invalid number
        then
                echo "There are invalid numbers in file. The program worked without considering$invalid."
        fi



else                                                                            # if file does not exist, write a warning and  exit.
        echo "$file: You must provide a filename as an argument"
        exit 1

fi
```

First, we ask the user to enter the filename for reading the file. **read file** part reads the argument from the user and puts it into the **file** variable.
Then, we check if the file exists or not. If it exists, we start to create the histogram. We create the **invalid** variable to hold numbers that are not in the 0-9 range so that the program continues to run even if there are invalid numbers in the file. The numbers to be displayed in the histogram are stored in the **countArray,** the program adds an * to the value in the index of the number for the numbers it reads in the file. After the reading of the file is finished, it shows the histogram and invalid numbers on the terminal.


Please, enter the file name.
file.txt
Histogram for file.txt.
0
1 **
2 ***
3
4 *
5 **
6 ***
7 *
8 *
9
MAIN MENU
Please select an option :
1. Create Histogram
2. Encryption
3. Delete oldest
4. Convert numbers
5. Organized files
6. Exit
Enter your choice [1-6] :

2- Encryption

```
if [[ $# -ne 2 ]]                               #exit if less than 2 argument
then
    echo "You must enter 2 arguments! "
    exit 1
fi

string=$1
key=$2
stringlength=${#1}
keylength=${#2}
encryptedText=""

if [[ $2 -lt 0 ]]                               # exit if number less or equal than 0
then
    echo "Second argument can not be a negative value! "
    exit 1
fi


if [[ $keylength -eq $stringlength ]]                       # if lenght of the string equals to the lenght of number
then
    for (( i = 0; i < $stringlength; i++)); do               # apply the increment char by char

        stringChar=${string:$i:1}
        keyChar=${key:$i:1}
        encryptedChar=`echo $stringChar| tr $(printf %${keyChar}s | tr ' ' '.')\a-zA-Z a-zA-Z`    # accept chars a-z and  A-Z convert between a-z and A-Z
        encryptedText=$encryptedText$encryptedChar                                              # save the changed string to text variable to show it on terminal
    done
    echo "$encryptedText"

elif [[ $keylength -eq 1 ]]                             # apple 1
then

 echo $string| tr $(printf %${key}s | tr ' ' '.')\a-zA-Z a-zA-Z            # if the lenght of number equals 1.

else

    echo "Key value lenght should be 1 or equal to the string length! "    #if lenght of key is not appropiate, exit.
        exit 1
fi
~
```

We read the arguments from the menu part for this program. First, we check the argument count. If the argument count is less than two, We will print an error into the terminal and exit the program.

Then we assign the first argument as **string**, the second argument as **key**(number), length of the string as **stringlength** and length of key as **keylength**. Then we assign a null string into the **encryptedText** variable.

If argument count is equal to two but the second argument is negative, we print an error message to the user and exit the program.

If there is no error about arguments, first we check the **keylength** and **stringlength** are equal. If they are equal, we traverse the string and key character by character. To do that, we defined two variables **stringChar** and **keyChar** . **stringChar** will be the current character in the string variable and **keyChar** will be in the key variable. Then we used the printf for ascii values of each character. **encryptedChar** variable will hold the encrypted version of each character. After we find the encrypted character, we will just append the **encryptedChar** into **encryptedText** variable. Last, after the for loop we just printed the encrypted string into the terminal.

Secondly, if the **keylength** is equal to one, we should encrypt the string by using only one value. It does the same thing with only one key value.

Lastly, we check the keylength is equal to 1 or string length. If it is not, we print an error message into terminal and exit the program.

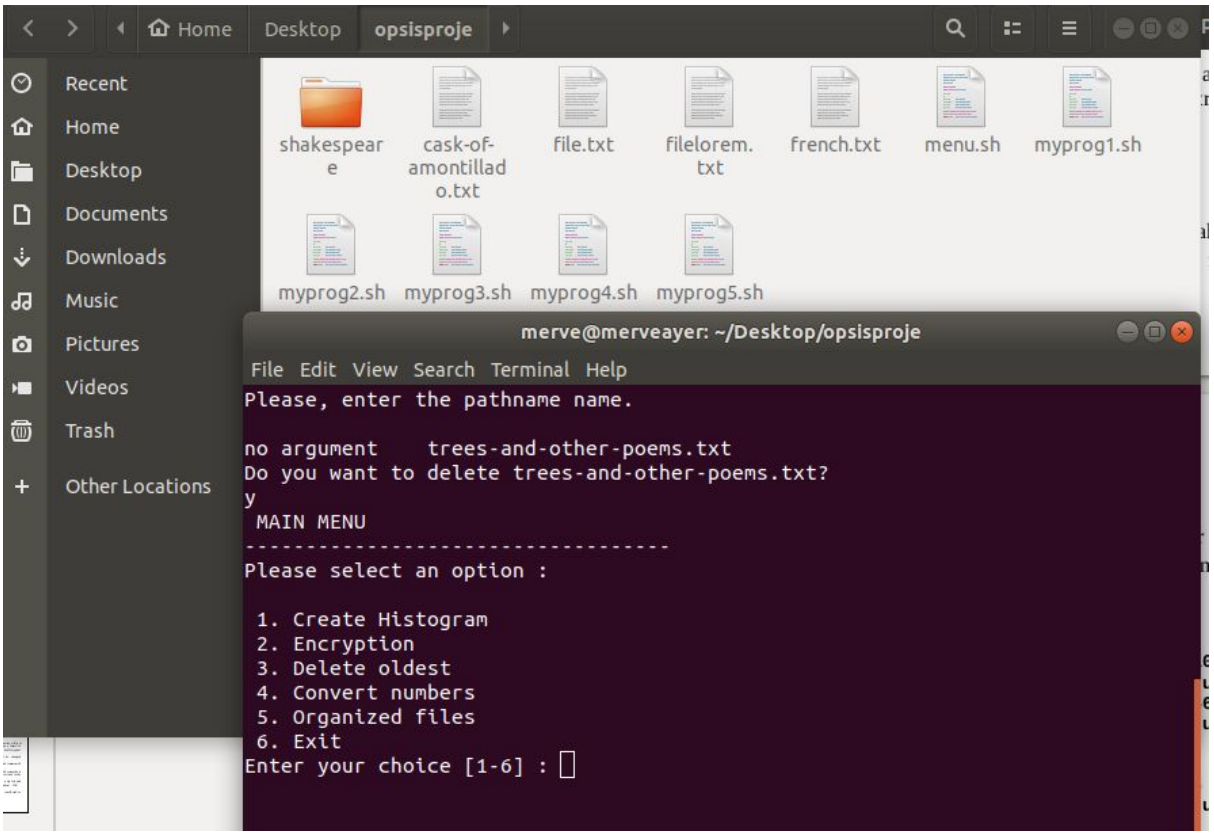
3-Delete Oldest

```
path=""
echo "Please, enter the pathname name."
read pathname

if [[ $# -eq 0 ]]                          #if there is no argument
then
    FILE_TO_DEL=$(ls -t | tail -n1)                  # list according to time/date, print the last one
    # be careful with this one
    path=$FILE_TO_DEL
    #rm -rf "$FILE_TO_DEL"
echo "no argument    $FILE_TO_DEL"


elif [[ $# -eq 1 ]]
then
    pathname=$1
    cd $pathname
    FILE_TO_DEL=$(ls -t | tail -n1)
    path=$FILE_TO_DEL

else
    echo “You can not enter 2 or more arguments!”
fi


echo "Do you want to delete $FILE_TO_DEL?"
read answer

if [[ $answer == "y" ]]
then
    rm -rf "$path"
elif [[ $answer == "n" ]]
then
    echo "no"
    exit 1
else
    echo "exit"
    exit 1
fi
```

First, we defined a null string **path** variable. Then ask the user for enter the pathname. If the pathname is null, the program will list the all files in the current directory and sorts them by time. Then the output of the **ls** part will be given into **tail** command. **tail -n1** command takes the last 10 files from the tail the first one which is the oldest file in the current directory. Then we assign this file into a **FILE_TO_DEL** variable and delete it with the **rm -rf** command. To do this first we ask the user want to delete this file or not. If the answer is no, program exits. If yes, then deletes the file.

If the pathname is given by the user, we changed the current directory into the given path. Then we do the same thing for the current directory.

```
merve@merveayer:~/Desktop/opsisproje$ ls -l
total 36
-rw-rw-r-- 1 merve merve    0 Kas 23 23:16 cask-of-amontillado.txt
-rw-rw-r-- 1 merve merve  397 Kas 23 23:10 filelorem.txt
-rw-rw-r-- 1 merve merve   26 Kas 23 22:41 file.txt
-rw-rw-r-- 1 merve merve    0 Kas 22 14:07 french.txt
-rwxrwxrwx 1 merve merve  872 Kas 22 15:53 menu.sh
-rwxrwxrwx 1 merve merve 1343 Kas 22 15:34 myprog1.sh
-rwxrwxrwx 1 merve merve 1335 Kas 22 17:00 myprog2.sh
-rwxrwxrwx 1 merve merve  741 Kas 22 15:48 myprog3.sh
-rwxrwxrwx 1 merve merve  528 Kas 23 22:44 myprog4.sh
-rwxrwxrwx 1 merve merve 1216 Kas 23 22:39 myprog5.sh
drwxrwxr-x 3 merve merve 4096 Kas 23 00:56 shakespeare
-rw-rw-r-- 1 merve merve    0 Kas 22 14:07 trees-and-other-poems.txt
merve@merveayer:~/Desktop/opsisproje$
```



4-Convert Numbers

```
echo "Please, enter the file name."
read file                                                       # get the file name.


stringValues=('zero' 'one' 'two' 'three' 'four' 'five' 'six' 'seven' 'eight' 'nine')

if [[ -n "$file" && `cat "$file"` ]];                           # check if the file exist
then
for (( i = 0 ; i <= 9; i++ )); do

search="$i"
char="${stringValues[$i]}"
sed -i "s/$search/$char/g" $file


done

cat $file

else                                                            # if file does not exist, write a warning and  exit.
        echo "$file: You must provide a filename as an argument"
        exit 1

fi
~
```

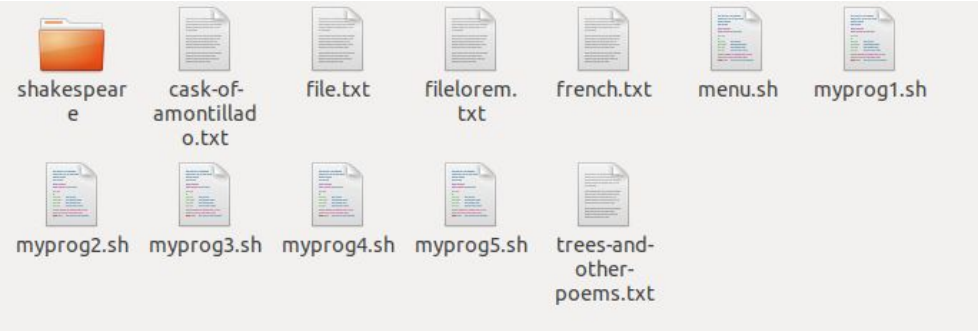
First we get the filename from the user. Then we defined an array name as **stringValues** and the elements of the **stringValues** array consist of numbers from zero to nine. If the file exists, then we will search a number inside the text file. If there is a number, **sed** command will find it and replace it with the corresponding string value in the **stringValues** array.
If the user enters a null pathname, program will give an error and exits.

```
Please, enter the file name.
filelorem.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit. seven Suspendissevitae
odio blandit, commodo nisl dignissim, nine commodo est. Quisqueblandit laoreet a
nte id tincidunt. Vivamus in vestibulum sem. Duis acfaucibus quam. Mauris posuer
e, sapien quis elementum porttitor, leoturpis finibus erat, vel dapibus zerozero
 lorem mauris in elit. Curabiturquis massa sit amet ligula suscipit pulvinar
 MAIN MENU
-----------------------------------
Please select an option :

 1. Create Histogram
 2. Encryption
 3. Delete oldest
 4. Convert numbers
 5. Organized files
 6. Exit
Enter your choice [1-6] :
```

5-Organized Files

```
echo "Please, write the  command. "
read -a command                                          # get the file name.



if [[ -n "$command" ]];                                      # check if the command is not null and exist. Agrument array lengt=2
then

    if [[ "${command[0]}" == "-R" ]]; then

        dName="${command[1]}"
        dName=${dName:1:-1}
        echo "$dName"
        FILES_TO_COPY=$(ls $dName)
        echo "$FILES_TO_COPY"
        destination_to_copy=($(ls -d */))
        len_des=$(ls -d */ | wc -l)
        echo "$destination_to_copy $len_des"

        ls -d */ | while read -r LINE
        do
          echo "$LINE"
          LINE=${LINE:0:-1}
          mkdir -p ./$LINE/copied
          cd /$LINE
          sudo cp $dname ./copied


       done




    else

        dName="${command[0]}"
        dName=${dName:1:-1}
        echo "$dName"
        FILES_TO_COPY=$(ls $dName)
        echo "$FILES_TO_COPY"
        mkdir -p ./copied
        sudo cp $dName ./copied


    fi


else                                              # if file does not exist, write a warning and  exit.
    echo "$file: You must provide a proper command as an argument"
    exit 1

fi
```

Program take arguments from user and check whether the argument start with -R or not. If it is, it cleans the quotation mark (“ ”) in the second argument to reach the wildcard. Then, it finds and saves files that match the wildcard to **FILES_TO_COPY** variable. It saves the subdirectories in the current directory to **destionation_to_copy** and the length number of subdirectories to **len_des**. Then, it creates copied files under subdirectories and saves FILES_TO_COPY into it. If there is no -R, it basically does the same thing but this time it will create only one copied file into the current directory.




```
Please, write the  command.
"c*.txt"
c*.txt
cask-of-amontillado.txt
[sudo] password for merve:
 MAIN MENU
---------------------------------
Please select an option :

 1. Create Histogram
 2. Encryption
 3. Delete oldest
 4. Convert numbers
 5. Organized files
 6. Exit
Enter your choice [1-6] :
```