30/10/2018

# CSE 3033 – Project 1 Report

Student 1: Mert Kelkit – 150115013
Student 2: Furkan Nakıp – 150115032

- In this project, we are supposed to handle 5 assignments given by course instructor. Explanations and outputs for each assignment goes below.

## Question 1:

In this question, we are supposed to take two arguments to our script – named as firstQuestion.sh -. First one is a string contains only lowercase letters, second one is a non-negative integer. Second argument must have the same length with the first argument or length 1.
Here are the input controls that we wrote:

```
10
11# $# gives the number of arguments given by user. Ensure that user gave exactly 2 arguments.
12if [ $# -ne 2 ]
13then
14    echo 'You have to give exactly 2 arguments'
15    echo 'Program exits...'
16    exit 1
17fi
18
19# Regular expression that matches only with positive integer
20re='^[0-9]+$'
21# If no match, print error
22if ! [[ $second_arg =~ $re ]] ; then
23    echo 'Second argument must be a nonnegative integer.'
24    echo 'Program exits...'
25    exit 1
26fi
27
28# First condition is first argument and second argument will have the same length
29# Second contition is second argument will have length 1 independent from first argument's length
30if [ ${#first_arg} -ne ${#second_arg} ]
31then
32    # If both conditions are not true, print error and exit
33    if [ ${#second_arg} -ne 1 ]
34    then
35        echo 'Second parameter should be one or has the same length with the first parameter.'
36        echo 'Program exits...'
37        exit 1
38    fi
39fi
40
41# Checks if each character of the first argument is a lowercase letter
42limit=${#first_arg}
43for (( i=0; i<limit; i++ ))
44do
45    # Even 1 character is not a lowercase letter, print error and exit
46    if [[ ! ${first_arg:i:1} =~ [a-z] ]] ; then
47        echo "First argument must contain only lower case letters"
48        echo "Program exits..."
49        exit 1
50    fi
51done
```

If inputs can pass these controls, we get ascii value of each character in the first argument in order to cipher them.

```
# Declare array of ciphered word
declare -a ciphered

# If second argument has the same length with the first argument
if [ ${#second_arg} -ne 1 ]
    then
    # Iterate through all characters of first argument
    for i in `seq 0 $((${#first_arg} - 1))`
    do
        # Get the ascii value of current character
        ascii_val=`echo -n ${first_arg:$i:1} | od -An -tuC`
        # Get the corresponding number which will be added to the character
        offset=${second_arg:$i:1}
        # Get ciphered ascii value
        new_ascii_val=$((ascii_val + offset))
        # If new ascii value comes after letter z ...
        if [ $new_ascii_val -ge 123 ]
        then
            # This operation finds actual ascii value for example: z + 1 = a, y + 5 = d
            new_ascii_val=$(( (new_ascii_val % 123) + 97 ))
        fi
        # Convert ascii value to its hex value in order to convert hex value to corresponding character
        hex_ascii_val=`printf "%x\n" $new_ascii_val`
        # Append character to the array
        ciphered[i]=`echo -e "\x$hex_ascii_val"`
    done
# If second argument has the length 1
else
    # Iterate through every character in the first argument
    for i in `seq 0 $((${#first_arg} - 1))`
    do
        # Get ascii value of the current character
        ascii_val=`echo -n ${first_arg:$i:1} | od -An -tuC`
        # Add second argument to the current character's ascii value
        new_ascii_val=$((ascii_val + second_arg))
        if [ $new_ascii_val -ge 123 ]
        then
            # If new ascii value comes after z, this operation rewinds ascii values as exampled above
            new_ascii_val=$(( (new_ascii_val % 123) + 97 ))
        fi
        # Get corresponding hex value
        hex_ascii_val=`printf "%x\n" $new_ascii_val`
        # Add ciphered character to the array
        ciphered[i]=`echo -e "\x$hex_ascii_val"`
    done
fi
```

We first declared an array called ‘ciphered’ which is empty at first. It will be filled according to the length of the second argument. After getting ascii values of each character, add corresponding value to the character’s ascii value. If new ascii value is larger than z’s ascii value, sum operation must rewind like z + 1 = a. Modulo and sum operations in the inner if condition provides that.
After filling ciphered array, we printed it to the screen.

30/10/2018

Sample outputs with menu:
* Menu calls ./firstQuestion.sh mert 1683

```
MAIN MENU
1. Cipher words
2. Create story
3. Move files
4. Convert hexadecimal
5. Delete files
6. Exit
Enter your choice: 1
Enter the first argument: mert
Enter the second argument: 1683

Ciphered word: nkzw
```

* Menu calls ./firstQuestion.sh furkan 8

```
MAIN MENU
1. Cipher words
2. Create story
3. Move files
4. Convert hexadecimal
5. Delete files
6. Exit
Enter your choice: 1
Enter the first argument: furkan
Enter the second argument: 8

Ciphered word: nczsiv
```

## Question 2:

In this question, we need to give only one argument which is the new file name to the script. Input control:

```
# Check argument number. If it's not exactly 1, print error and exit
if [ $# -ne 1 ]
then
    echo 'You have to give 1 argument as file name'
    echo 'Program exits...'
    exit 1
fi
```

After passing this condition, script checks for existence of given file name. If file exists, script asks the user that he/she wants to overwrite the file or not.
Input controls handled below if user doesn't enter y or n.
If file doesn't exists in current working directory, script passes through this condition.

```
# If given file name is found
if [ -f $file_name ]; then
    # Prompt user to overwrite it or not
    read -r -p "File found! Do you want it to be overwritten? (y/n): " y_or_n
    # If no, exit program
    if [ "$y_or_n" = n ]; then
        echo "Program exits. Bye !"
        exit 0
    fi
    # If user didn't enter a valid choice, warn the user until he/she enters a valid choice.
    while [ "$y_or_n" != n ] && [ "$y_or_n" != y ]; do
        read -r -p "Please enter y or n: " y_or_n
        if [ "$y_or_n" = n ]; then
            echo "Program exits. Bye !"
            exit 0
        # If yes, program will overwrite to the given file name.
        elif [ "$y_or_n" = y ]; then
            break
        fi
    done
fi
```

After above code block, we created an array contains input file names. We tried to open these input files, if files exist. If a file doesn't exist, script gives error and exits the program.
After opening a file, split file context line by line. Each line stored in a temporary array, which will used for choosing a random line.
We know how many lines file contains, by the counter variable. Then append selected random line to the array 'lines'. After that, print selected lines to the given file name.

```
39 # These are the input files, we will get one lines from each of them.
40 inputs=("giris.txt" "gelisme.txt" "sonuc.txt")
41 declare -a lines
42
43 # Iterate through each input file
44 for i in `seq 0 $((${#inputs[@]} - 1))`
45 do
46     # If any input files cannot be found, print error and exit
47     if [ ! -f "${inputs[i]}" ]; then
48         echo "${inputs[i]} cannot found!"
49         echo "Program exits..."
50         exit 1
51     fi
52
53     # If file exists, read file line by line (IFS determines seperator which is newline (\n))
54     declare -a temp_lines
55     IFS=$'\n'
56     counter=0
57     # Read all file
58     for next in `cat ${inputs[i]}`
59     do
60          # Append each line to the temporary array
61          temp_lines[counter]="$next"
62          # Counts number of lines(paragraphs)
63          ((counter++))
64     done
65     # Choose a random element from the array - which corresponds to a line.
66     # Inner operation returns a random number between 0 and (line_count-1).
67     lines[i]=${temp_lines[$((RANDOM % counter))]}
68 done
69
70 # Output the chosen lines to the given file name. Each line seperated with \n\n for a good appearance
71 printf "%s\n\n" "${lines[@]}" > $file_name
72
73 echo "A random story is created to the file ${file_name}"
```

Sample executions goes below – executed with menu – :

```
mertk@mertk:~/eclipse-workspace/opsis$ ls -l
total 48
drwxr-xr-x 3 mertk mertk 4096 Eki 30 20:47 asdf
-rw-r--r-- 1 mertk mertk    0 Eki 30 20:47 ferhat.txt
-rwxr-xr-x 1 mertk mertk 2974 Eki 30 21:16 fifthQuestion.sh
-rwxr-xr-x 1 mertk mertk 3107 Eki 30 20:58 firstQuestion.sh
-rwxr-xr-x 1 mertk mertk 1932 Eki 30 20:47 fourthQuestion.sh
-rw-r--r-- 1 mertk mertk  496 Eki 30 20:47 gelisme.txt
-rw-r--r-- 1 mertk mertk  268 Eki 30 20:47 giris.txt
-rwxr-xr-x 1 mertk mertk 2583 Eki 30 21:23 menu.sh
-rw-r--r-- 1 mertk mertk  360 Eki 30 20:47 newfile.txt
-rwxr-xr-x 1 mertk mertk 2120 Eki 30 21:04 secondQuestion.sh
-rw-r--r-- 1 mertk mertk  307 Eki 30 20:47 sonuc.txt
-rw-r--r-- 1 mertk mertk  231 Eki 30 20:47 story.txt
-rwxr-xr-x 1 mertk mertk 1255 Eki 30 21:08 thirdQuestion.sh
```

There is a file called “newfile.txt” as we can see.

```
MAIN MENU
1. Cipher words
2. Create story
3. Move files
4. Convert hexadecimal
5. Delete files
6. Exit
Enter your choice: 2
Enter the first argument: newfile.txt

File found! Do you want it to be overwritten? (y/n): y
A random story is created to the file newfile.txt
```

Here is the context of newfile.txt :

```
mertk@mertk:~/eclipse-workspace/opsis$ cat newfile.txt
Buralardan uzakta kucuk bir ulke varmis.

Yardimsever arkadasim hemen imdadina kostu. Dikkatlice kozanin liflerini siyirdi, kozayi araladi ve kelebegin fazla cabalamadan kozadan cikmasini sagladi.

Arkadasim kelebegin isini kolaylastirmak isterken onun guclenmesine engel olmustu.
```

## Question 3:

In this question, we don’t need to provide any arguments. Script moves all files which have write permission for users to a directory called “writable”. Script first checks is there any directory called “writable”. If there isn’t, script creates directory “writable”.

```
# Create directory "writable" if it's absent.
if [ ! -d "writable" ]; then
    mkdir writable
fi

# Array to store all writable files by user
declare -a writable_files

# Execute a find command which with a maxdepth 1 - means that look for only current directory. Type f means look for only files, not directories.
# perm -u+w means look for files which are writable for users. print0 for null escape characters
# This loop iterates through each line of this "find" command's response.
while IFS=  read -r -d $'\0'; do
    # Append each line to the writable files array
    writable_files+=("$REPLY")
done < <(find . -maxdepth 1 -type f -perm -u+w -print0)

# Get the number of files
no_of_files=${#writable_files[@]}

# Iterate through all file names
for i in `seq 0 $((no_of_files - 1))`
do
    # Move given file to the given directory
    # If a file name contains space...
    if [ `echo ${writable_files[i]} | grep \  | wc -l` -ne 0 ]; then
        # File name must written between double quotes in order to prevent errors.
        mv "${writable_files[i]}" writable
    else
        mv ${writable_files[i]} writable
    fi
done

# Information message
echo "$no_of_files files are moved to the writable directory."
```

While loop iterates through returning value of find command – which is like an array of file names – . -maxdepth 1 option means look for current directory. -type f means look only for files. - perm -u+w means list files which have write permission for users. There must be a check for file names, file name contains any spaces or not. Expression written inside grep determines is there any white space in the file name. If there are any white spaces in the file name, mv command calls file name inside of double quotes. If there is no space, mv command calls file name directly, no need for double quotes.

Sample execution with menu:

```
MAIN MENU
1. Cipher words
2. Create story
3. Move files
4. Convert hexadecimal
5. Delete files
6. Exit
Enter your choice: 3

13 files are moved to the writable directory.
*Current directory /home/mertk/eclipse-workspace/opsis changed to /home/mertk/eclipse-workspace/opsis/writable
```

Sample execution with ./thirdQuestion.sh

```
mertk@mertk:~/Desktop/test$ ls -l
total 4
-r-xrwxrwx 1 mertk mertk   0 Eki 30 04:01  shakespeare_nonwritable.txt
-rw-r--r-- 1 mertk mertk   0 Eki 30 04:01  shakespeare_second.txt
-rw-r--r-- 1 mertk mertk   0 Eki 30 04:00  shakespeare.txt
-rw-r--r-- 1 mertk mertk   0 Eki 30 04:16 'shakespeare with  many space.txt'
-rw-r--r-- 1 mertk mertk   0 Eki 30 04:01 'shakespeare with space.txt'
-rwxr-xr-x 1 mertk mertk 561 Eki 30 16:35  thirdQuestion.sh
mertk@mertk:~/Desktop/test$ ./thirdQuestion.sh
5 files are moved to the writable directory.
mertk@mertk:~/Desktop/test$ ls -l
total 4
-r-xrwxrwx 1 mertk mertk    0 Eki 30 04:01 shakespeare_nonwritable.txt
drwxr-xr-x 2 mertk mertk 4096 Eki 30 22:12 writable
mertk@mertk:~/Desktop/test$ cd writable
mertk@mertk:~/Desktop/test/writable$ ls -l
total 4
-rw-r--r-- 1 mertk mertk   0 Eki 30 04:01  shakespeare_second.txt
-rw-r--r-- 1 mertk mertk   0 Eki 30 04:00  shakespeare.txt
-rw-r--r-- 1 mertk mertk   0 Eki 30 04:16 'shakespeare with  many space.txt'
-rw-r--r-- 1 mertk mertk   0 Eki 30 04:01 'shakespeare with space.txt'
-rwxr-xr-x 1 mertk mertk 561 Eki 30 16:35  thirdQuestion.sh
mertk@mertk:~/Desktop/test/writable$
```

As we can see, all files with valid permissions moved to the writable directory. Files that doesn’t have write permission for users are not moved to the writable directory.

## Question 4:

In this question, we need to provide an argument. Then we need to print all prime numbers less than given argument with hexadecimal representation.
We wrote a function called isprime. This function checks if a number is prime or not prime. If given number is prime, it returns 1. If not, it returns 0.

Implementation of function isprime:

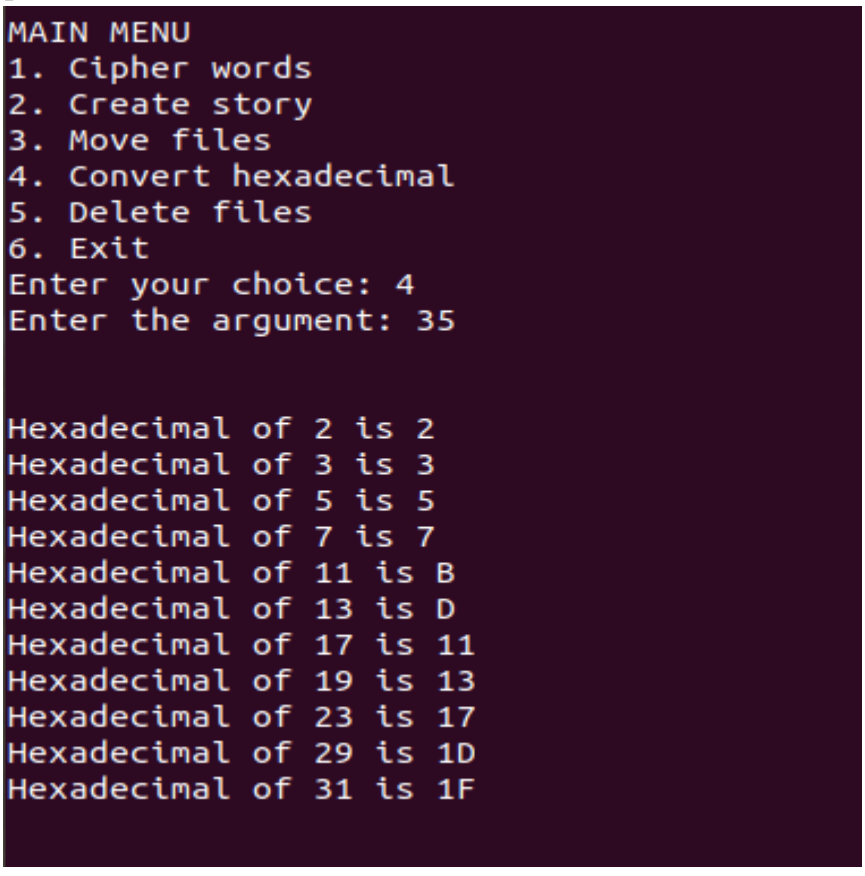
```
# This is a function to check an integer's primeness
# If integer is prime, returns 1
# If integer is not prime, returns 0
function isprime {
    # Get the first argument - number to be checked
    local var=$1

    # If number is 1, 1 is not prime by definition
    if [ "$var" -eq 1 ]; then
        return 0
    fi

    # If number is 2, 2 is prime by definition
    if [ "$var" -eq 2 ]; then
        return 1
    fi

    # Get the possible maximum divider of the number. Square root of the number is the largest possible divider from number theory
    maximum_divider=$(echo "sqrt($var)" | bc -l)
    # Get the ceil of the square rooted number - most probably it'll be a floating number
    maximum_divider=${maximum_divider%.*}

    # Start possible dividers with 2, increment each step, stop when it reaches to the maximum possible driver
    for (( j=2; j<=$maximum_divider; j++ )); do
        # If number is divided by divider, it means it's a prime number. Return 1
        if [ $(( $var % $j )) -eq 0 ]; then
            return 0
        fi
    done
    # If none of
    return 1
}
```

Function tries to divide given number with all possible dividers. If any of them can divide number, function will return 0. This function will be used later in the script.

On the next page, first script controls argument number. It must be one, if not, program exits. After that, program need to check given argument is a non-negative integer or not. This control made by a simple regular expression. If argument doesn’t match, program exits.
After that, program checks inertness of each number less than given argument with isprime function. If function returns 1, program prints hexadecimal representation of prime number with printf command (%X means upper case hexadecimal representation).

```
39 # Get the first argument
40 number=$1
41
42 # If there is not only 1 argument, print error and exit
43 if [ $# -ne 1 ]
44 then
45     echo 'You have to give exactly 1 integer argument'
46     echo 'Program exits...'
47     exit 1
48 fi
49
50 # This is the regular expression, matches with only positive integers
51 re='^[0-9]+$'
52 # If there is no match(argument is not positive integer), print error and exit
53 if ! [[ $number =~ $re ]] ; then
54     echo 'You must enter a nonnegative integer.'
55     echo 'Program exits...'
56     exit 1
57 fi
58
59 # Test all numbers which are less than argument
60 for (( i=1; i<$number; i++ )); do
61     isprime $i
62     # Get the last returning value from last function call, which is isprime
63     is_prime=$?
64     # If current number (i) is prime, print it's hexadecimal representation
65     if [ "$is_prime" -eq 1 ]; then
66         hex_val=`printf "%X\n" $i`
67         echo "Hexadecimal of $i is $hex_val"
68     fi
69 done
```

Sample output executed with menu:



## Question 5:

In this question, we need to provide one or two arguments. First one is a wildcard which is mandatory; second one is path name which is optional.
Here are the input controls:

```
# Get arguments. First argument (wildcard) is mandatory, second (path name) is optional.
first_arg=$1
second_arg=$2

# If there is not 1 argument...
if [ "$#" -ne 1 ]; then
    # ...also there is not 2 arguments...
    if [ "$#" -ne 2 ]; then
        # Print error message and exit. There must be one or two arguments.
        echo "You must provide a wildcard as first argument. Pathname argument is optional."
        echo "Program exits..."
        exit 1
    fi
    # If second argument is given, but this path name doesn't exist in current working directory, print error message and exit.
    if [ ! -d $second_arg ]; then
        echo "No directory found named $second_arg"
        echo "Program exits..."
        exit 1
    fi
fi

# Wildcards in bash scripting language.
wildcards=( '*' '.' '?' '|' ']' '[' )
wildcard_counter=0
# Iterate through all wildcard characters
for wildcard in "${wildcards[@]}"
do
  if [[ $first_arg == *"${wildcard}"* ]]; then
    # Increment counter if there is a wildcard found in the first argument.
    ((wildcard_counter++))
  fi
done

# If counter is never incremented...
if [ "$wildcard_counter" -eq 0 ]; then
      # Print corresponding error message and exit. First argument doesn't include any wildcards.
      echo "There is no wildcard found in the first argument."
      echo "Program exits..."
      exit 1
fi
```

First control is for number of arguments. There can be 1 or 2 arguments. If user provided second argument as a path name, there will be one more control. Script checks that given path name exists or not. If not, program exits.
Then second control is about first argument. For loop iterates through all characters of first argument. If there is any wildcard in the first argument, counter is incremented. After checking all characters if counter is 0 – that means given argument is not wildcard – program prints error and exits.

```
# If there is only one argument - just wildcard -, look for matching files with find command in the current working directory.
# -maxdepth 1 means just look for current working directory.
# -type f means just look for files
# -name "$first_arg" means look for files which their names matches with the wildcard argument.
if [ "$#" -eq 1 ]; then
    found_files=($(find . -maxdepth 1 -type f -name "$first_arg"))
# If second argument is given...
else
    # Look for every file matches with wildcard in the "path_name" directory.
    #There is no -maxdepth option because we want to look for every file in its subdirectories.
    found_files=($(find ./$second_arg -type f -name "$first_arg"))
fi

# If no file is found, print error message and exit
if [ ${#found_files[@]} -eq 0 ]; then
    echo "No files found matching with wildcard."
    exit 0
fi

# Iterate through every found files.
for f in "${found_files[@]}"
do
    # Prompt user to delete current file or not
    read -r -p "Do you want to delete $f? (y/n): " y_or_n
    # If yes, remove current file
    if [ "$y_or_n" = y ]; then
        rm $f
        echo "$f removed."
    # If no, continue with other files if exist.
    elif [ "$y_or_n" = n ]; then
        continue
    # If user doesn't enter a valid option...
    else
        # Until user enters a valid option...
        while [ "$y_or_n" != n ] && [ "$y_or_n" != y ]
        do
            # Warn the user and get input again
            read -r -p "Please enter y or n: " y_or_n
            # If yes, remove file
            if [ "$y_or_n" = y ]; then
                rm $f
                echo "$f removed."
            # If no, continue with other files if exist.
            elif [ "$y_or_n" = n ]; then
                break
            fi
        done
    fi
done
```

There are two cases for usage of command “find”.

- First one is for only one wildcard argument given:
  In this mode, script looks only for files which match with given wildcard only in current working directory.

- Second one is for both arguments given:
  In this mode, script looks for files which match with given wildcard in given directory with second argument and all of its sub directories. We did not provide -maxdepth option in order to make that operation, looks for all sub directories.

If found files array has the length 0, it means that no files can be found which match with given wildcard.

If any files found, script iterates through all file names in the array and asks user that he/she wants to delete the file. User must enter y or n as a choice. If user didn't enter these values, program will warn user and tries to get input again.
If user enters y, program will remove that file. If user enters n, program will do nothing.

Directories under the present working directory before execution:

```
mertk@mertk:~/eclipse-workspace/opsis$ ls -l
total 48
drwxr-xr-x 3 mertk mertk 4096 Eki 30 20:47 asdf
-rw-r--r-- 1 mertk mertk    0 Eki 30 20:47 ferhat.txt
-rwxr-xr-x 1 mertk mertk 2974 Eki 30 21:16 fifthQuestion.sh
-rwxr-xr-x 1 mertk mertk 3107 Eki 30 20:58 firstQuestion.sh
-rwxr-xr-x 1 mertk mertk 1932 Eki 30 20:47 fourthQuestion.sh
-rw-r--r-- 1 mertk mertk  496 Eki 30 20:47 gelisme.txt
-rw-r--r-- 1 mertk mertk  268 Eki 30 20:47 giris.txt
-rwxr-xr-x 1 mertk mertk 2583 Eki 30 21:23 menu.sh
-rw-r--r-- 1 mertk mertk  360 Eki 30 20:47 newfile.txt
-rwxr-xr-x 1 mertk mertk 2120 Eki 30 21:04 secondQuestion.sh
-rw-r--r-- 1 mertk mertk  307 Eki 30 20:47 sonuc.txt
-rw-r--r-- 1 mertk mertk  231 Eki 30 20:47 story.txt
-rwxr-xr-x 1 mertk mertk 1255 Eki 30 21:08 thirdQuestion.sh
```

```
mertk@mertk:~/eclipse-workspace/opsis/asdf$ ls -l
total 4
drwxr-xr-x 2 mertk mertk 4096 Eki 30 20:47 abc
-rw-r--r-- 1 mertk mertk    0 Eki 30 20:47 ferhat.txt
```

```
mertk@mertk:~/eclipse-workspace/opsis/asdf/abc$ ls -l
total 0
-rw-r--r-- 1 mertk mertk 0 Eki 30 20:47 firstfile.txt
```

Sample output launched on menu:

```
MAIN MENU
1. Cipher words
2. Create story
3. Move files
4. Convert hexadecimal
5. Delete files
6. Exit
Enter your choice: 5
Enter the first argument: f*
Enter the second argument(Optional, if you dont want to give, press enter):

Do you want to delete ./firstQuestion.sh? (y/n): n
Do you want to delete ./fifthQuestion.sh? (y/n): n
Do you want to delete ./ferhat.txt? (y/n): y
./ferhat.txt removed.
Do you want to delete ./fourthQuestion.sh? (y/n): n
```

After execution:

```
mertk@mertk:~/eclipse-workspace/opsis$ ls -l
total 48
drwxr-xr-x 3 mertk mertk 4096 Eki 30 20:47 asdf
-rwxr-xr-x 1 mertk mertk 2974 Eki 30 21:16 fifthQuestion.sh
-rwxr-xr-x 1 mertk mertk 3107 Eki 30 20:58 firstQuestion.sh
-rwxr-xr-x 1 mertk mertk 1932 Eki 30 20:47 fourthQuestion.sh
-rw-r--r-- 1 mertk mertk  496 Eki 30 20:47 gelisme.txt
-rw-r--r-- 1 mertk mertk  268 Eki 30 20:47 giris.txt
-rwxr-xr-x 1 mertk mertk 2583 Eki 30 21:23 menu.sh
-rw-r--r-- 1 mertk mertk  282 Eki 30 22:02 newfile.txt
-rwxr-xr-x 1 mertk mertk 2120 Eki 30 21:04 secondQuestion.sh
-rw-r--r-- 1 mertk mertk  307 Eki 30 20:47 sonuc.txt
-rw-r--r-- 1 mertk mertk  231 Eki 30 20:47 story.txt
-rwxr-xr-x 1 mertk mertk 1255 Eki 30 21:08 thirdQuestion.sh
```

As we can see above, ferhat.txt removed after execution.

Sample execution with two arguments given is on the next page.

```
MAIN MENU
1. Cipher words
2. Create story
3. Move files
4. Convert hexadecimal
5. Delete files
6. Exit
Enter your choice: 5
Enter the first argument: f*
Enter the second argument(Optional, if you dont want to give, press enter): asdf

Do you want to delete ./asdf/ferhat.txt? (y/n): y
./asdf/ferhat.txt removed.
Do you want to delete ./asdf/abc/firstfile.txt? (y/n): y
./asdf/abc/firstfile.txt removed.
```

Now, list of directory “asdf” and its sub directory “abc”:

```
mertk@mertk:~/eclipse-workspace/opsis/asdf$ ls -l
total 4
drwxr-xr-x 2 mertk mertk 4096 Eki 30 22:10 abc
mertk@mertk:~/eclipse-workspace/opsis/asdf$
```

```
mertk@mertk:~/eclipse-workspace/opsis/asdf/abc$ ls -l
total 0
mertk@mertk:~/eclipse-workspace/opsis/asdf/abc$
```

As expected, files starts with f are all removed.

## Bonus - Menu:

We implemented menu as bonus assignment. Here is the implementation:

```
clear

# INFINITE LOOP
while :
do
    # Print menu
    echo "MAIN MENU"
    echo "1. Cipher words"
    echo "2. Create story"
    echo "3. Move files"
    echo "4. Convert hexadecimal"
    echo "5. Delete files"
    echo "6. Exit"
    # Get user's choice
    read -r -p "Enter your choice: " choice
```

In this part, we printed menu choices and waited for a user input. According to given input, there is a case statement which helps us to execute which script when user enters a choice.

Here are the statements:

```
    case "$choice" in
    # If user entered 1, get required arguments from the user. There is no argument control because our script will control the arguments.
    1)  read -r -p "Enter the first argument: " first_arg
        read -r -p "Enter the second argument: " second_arg
        echo
        # Run script with given arguments.
        ./firstQuestion.sh $first_arg $second_arg
        echo
        echo
        ;;
    # If user entered 2, get required file name argument. Script controls the argument
    2)  read -r -p "Enter the first argument: " first_arg
        echo
        # Run second question's script.
        ./secondQuestion.sh $first_arg
        echo
        echo
        ;;
    # If user entered 3...
    3)  prev_dir=$PWD               # Store current directory. Third script will change current working directory because of it is a writable file too.
        echo
        # Run script
        ./thirdQuestion.sh
        # Change directory in order to continue executing scripts, because they are all writable files for users.
        cd ./writable
        # Print user that current working directory is changed.
        echo "*Current directory $prev_dir changed to $PWD"
        echo
        echo
        ;;
    # If user entered 4, get the first argument. Its control will be done in the script.
    4)  read -r -p "Enter the argument: " first_arg
        echo
        echo
        # Run script with it's argument.
        ./fourthQuestion.sh $first_arg
        echo
        echo
        ;;
    # If user entered 5, get the first and second arguments. First one is mandatory, second one is optional.
    5)  read -r -p "Enter the first argument: " first_arg
        # If it's empty, it means that user did not give the second argument.
        read -r -p "Enter the second argument(Optional, if you dont want to give, press enter): " second_arg
        echo
        # If second argument is not given
        if [ ${#second_arg} -eq 0 ]; then
            # Run script with only first argument
            ./fifthQuestion.sh "$first_arg"
        # If second argument is given
        else
            # Run script with both arguments
            ./fifthQuestion.sh "$first_arg" $second_arg
        fi
        echo
        echo
        ;;
    # If user wants to exit, print information message and exit the program
    6)  echo "Program exits. Bye !"
        exit 0
        ;;
    # If user did not enter a valid choice...
    *)  echo
        echo
        # Warn user to enter a number between 1 and 6.
        echo "Please enter a number between 1 and 6."
        echo
        echo
        ;;
    esac
```

Script asks for arguments if chosen script requires any arguments.
Some of the screenshots above are from menu execution.
Here is appearance of menu which expects a user choice:

```
MAIN MENU
1. Cipher words
2. Create story
3. Move files
4. Convert hexadecimal
5. Delete files
6. Exit
Enter your choice: abc


Please enter a number between 1 and 6.


MAIN MENU
1. Cipher words
2. Create story
3. Move files
4. Convert hexadecimal
5. Delete files
6. Exit
Enter your choice:
```

When user presses exit, 6th choice:

```
MAIN MENU
1. Cipher words
2. Create story
3. Move files
4. Convert hexadecimal
5. Delete files
6. Exit
Enter your choice: 6
Program exits. Bye !
mertk@mertk:~/eclipse-workspace/opsis$
```

30/10/2018

References:

[1] https://stackoverflow.com/
[2] https://stackexchange.com/
[3] https://ryanstutorials.net/bash-scripting-tutorial/
[4] http://akademik.marmara.edu.tr/zuhal.altuntas/teaching