CSE 333 - OPERATING SYSTEMS
Programming Assignment # 1
Marmara University - Computer Engineering

Akif Batur - 150111854

1.
$1: First Argument
$2: Second Argument
Usage: arg1 (number of files) [arg2] (directory)

Error checking:
a) Check the first argument of the program whether it's exist or not.
b) Check the first argument of the program whether it's greater than zero or not.
c) Check the first argument of the program whether it's an integer or not. If it is contains other than a numerical value, the program will halt.
d) If the second argument of the program is provided then check that directory whether it's exist or not.

Step by step main code: ls -al $2 | grep ^- | awk '{print $5,$9}' | sort -n -r | head -n $1
a) ls -al $2:
List files and directories under the directory $2. if $2 is not given, list the current directory
b) ls -al $2 | grep ^- :
List files
only via using grep
c) ls -al $2 | grep ^- | awk '{print $5,$9}':
List the files with the size and name only
. Re-format the output via using awk.
awk: pattern scanning and processing language
$5: File size
$9: File name
d) ls -al $2 | grep ^- | awk '{print $5,$9}' | sort -n -r:
List files with the size and name only in numerical order and then reverse the order
e) ls -al $2 | grep ^- | awk '{print $5,$9}' | sort -n -r | head -n $1:
Print the first $1 of lines

Example output:
aqui@aqui:~$ ./q1.sh 10
14848 report.doc
1994 q4.sh
1197 q3.sh
1196 q1.sh
1020 q6.sh
542 q2.sh
444 q5.sh

2.
$1: First Argument
$2: Second Argument
Usage: arg1 (pattern)

Error checking:
a) Check the first argument of the program whether it's exist or not.

Step by step main code: find -iname "*$1*" -type f -printf "%TY-%Tm-%Td %TH:%TM %f\n" | sort -n | awk '{print "The file " $3 " was modified on " $1, "at " $2"."}'
a) find -iname "*$1*" -type f -printf "%TY-%Tm-%Td %TH:%TM %f\n":
-iname pattern
: Like -name, but the match is case insensitive.
-type f: File is of type regular file.
-printf "%TY-%Tm-%Td %TH:%TM %f\n": print format on the standard output, interpreting '\' escapes and '%' directives. Field widths and precisions can be
specified as with the 'printf' C function.
%TY: year
%Tm: month (01..12)
%Td: day of month (01..31)
%TH: hour (00..23)
%TM: minute (00..59)
%f: File name
b) find -iname "*$1*" -type f -printf "%TY-%Tm-%Td %TH:%TM %f\n" | sort -n
Numerical sort according to date and hour
c) find -iname "*$1*" -type f -printf "%TY-%Tm-%Td %TH:%TM %f\n" | sort -n | awk '{print "The file " $3 " was modified on " $1, "at " $2"."}'
Re-format the output via using awk.
$1: Modification date of the file
$2: Modification hour of the file
$3: File name

Example output:
aqui@aqui:~$ ./q2.sh .sh
The file q6.sh was modified on 2014-11-02 at 16:29.
The file q2.sh was modified on 2014-11-02 at 16:30.
The file q3.sh was modified on 2014-11-02 at 16:34.
The file q5.sh was modified on 2014-11-04 at 13:06.
The file q4.sh was modified on 2014-11-04 at 13:10.
The file q1.sh was modified on 2014-11-05 at 11:45.

3.
$1: First Argument
$2: Second Argument
Usage: arg1 (pattern) [arg2] (directory)

Error checking:
a) First check if the temproraryfile.txt is exist or not in the current directory.
b) Check the first argument of the program whether it's exist or not.

Step by step main code:
a)
maindirectory=$(pwd)"/*"
Get the current directory.
directory=$maindirectory
directory=$2"/*"

If the second argument which is the directory is not given then set it to current directory, otherwise set it to the given directory.
mainfile=$(pwd)"/temporaryfile.txt"
This file will be used to cat the raw output. Then we will read and sort the words from this file.
b) Find all files under the directory with find command in a for loop
c) If the current processing file is the temprorayfile.txt then stop processing and continue to the next file.
d) Start to reading each file line by line in a while loop. We need to read line by line because there may be more than one word in each line.
e) Get the each word in the each line in a for loop and check if it is contains the pattern. If it's so, then write it to the temproraryfile.txt
f) After processing all files and writing the correct words to the temproraryfile.txt;
cat temporaryfile.txt | sort | grep -v '\/'**| column -c 80**
cat temporaryfile.txt: Read temproraryfile.txt
cat temporaryfile.txt | sort: Sort the words
cat temporaryfile.txt | sort | grep -v '\/'**:** Invert the sense of matching, to select non-matching lines
cat temporaryfile.txt | sort | grep -v '\/'**| column -c 80:** Seperate output in columns
rm temporaryfile.txt: Remove temproraryfile.txt

Example output:
aqui@aqui:~$ ./q3.sh a*rum*
/home/aqui/klasor
Please wait...
/home/aqui/klasor/dict is processing...
/home/aqui/klasor/evening-sun.txt is processing...
/home/aqui/klasor/morning-sun.txt is processing...
/home/aqui/klasor/sun1.txt is processing...
/home/aqui/klasor/sun2.txt is processing...
Done!

RESULT

Agarum	androphorum	antistrumous	Arrhenatherum	aurum
alabastrum	anoestrum	antrum	Arum	autoserum
alarum	antirumor	Aplectrum	arumin	
ambulacrum	antiserum	archicerebrum	Asarum	
Anatherum	antistrumatic	arcocentrum	Ascyrum	

aqui@aqui:~/klasor$ ./q3.sh food
Please wait...
/home/aqui/klasor/dict is processing...
/home/aqui/klasor/evening-sun.txt is processing...
/home/aqui/klasor/morning-sun.txt is processing...
/home/aqui/klasor/q3.sh is processing...
/home/aqui/klasor/sun1.txt is processing...
/home/aqui/klasor/sun2.txt is processing...
Done!

RESULT

food	foodless	foodstuff	nonfood
foodful	foodlessness	foody	unfoodful

4.

$1: First Argument
$2: Second Argument
$3: Third Argument
Usage: arg1 (START) [arg2] (STOP) [arg3] (STEP)

Error checking:
a) If there are no arguments, then exit.
b) If there are more than 3 arguments are given, then exit.
c) START, STOP and STEP values must be an integer. If they are not, then exit.
d) STEP must be a positive integer. If it's not, then exit.

Step by step main code:
a)

- If there is one argument is given, then STOP is the first argument. START and STEP should be 1.
- If there are two arguments are given, then START is the first argument, STOP is the second argument and STEP is 1.
- If there are three arguments are given, then START is the first argument, STOP is the second argument and STEP is the third argument.

b) Now we have two possible situations:

- If STOP is smaller than START subtract STEP in a while loop but first echo out the START value at each iteration.
- If STOP is greater than START add STEP in a while loop but first echo out the START value at each iteration.

Example output:
aqui@aqui:~$ ./q4.sh 10 0 2
10
8
6
4
2
aqui@aqui:~$ ./q4.sh 0 10 2
0
2
4
6
8
aqui@aqui:~$ ./q4.sh 5
1
2
3
4
aqui@aqui:~$ ./q4.sh 5 0
5
4
3
2
1

5.
Usage: We can run the script individually and enter the numbers to find the total numbers or we can pipe the result of the q4.sh.

Error checking:
a) Each input must be an integer otherwise it will be not added to the total.

Step by step main code:
a) Start an infinite loop.
b) Read a number.
c) In a switch-case structure, check each number. If it's an integer, then add it to the total otherwise do nothing.
d) If the input is an empty line, then echo out the total and exit.

Example output:
aqui@aqui:~$ ./q5.sh
1
2
3
4
5

Total is 15.
aqui@aqui:~$ ./q4.sh 1 6 1 | ./q5.sh
Total is 15.

6.
Usage: [arg1] (pattern) [arg2]...[arg(n)] (directories)

Error checking:
a) If any of the given directories are not exist then program should exit.
b) If any file not found with a given pattern, then total number of the line is must be zero.
c) grep command will help us to count number of lines starting with a character. So we do not need an extra effort to check for EOF. grep will give us the correct result at each time even if there are empty lines between any of two lines. If the empty line is located at the end of file, grep will handle that as we said before.

Step by step main code:
There are three situations depending on given arguments
a) If there is no argument, then we should process all the files recursively and echo out the total numbers.
for f in $(find . -type f): This will help us to find and examine all files recursively in the current directory which is denoted by "**.**".
grep -c ^ $f: This will give us to get the correct total number of the lines. $f is the current file name with its path.
b) If there is one argument is given, then it's the pattern. So we should examine all the files with the given pattern recursively.
for f in $(find . -type f -iname "*$1*"): This will help us to find files with the given pattern in the current directory which is denoted by "**.**".
grep -c ^ $f: Same as part a.
c) If there are more than one argument is given, then we should examine recursively in the directories as given after the first argument, for all the the files with the given pattern as the first argument.
for f in $(find ${*:2} -type f -iname "*$1*"): This will help us to find files with the given pattern.
${*:2}: This means arguments after the first argument. Will be used for directories.

grep -c ^ $f: Same as part a.

Example output:

```
aqui@aqui:~$ tree courses/
courses/
├── cse141
│   └── ders
│       └── test1.java
├── cse142
│   └── test2.java
├── klasor
│   ├── altklasor
│   │   ├── dict
│   │   └── evening-sun.txt
│   ├── morning-sun.txt
│   ├── sun1.txt
│   └── sun2.txt
└── q6.sh
5 directories, 8 files
aqui@aqui:~$ cd courses/
aqui@aqui:~/courses$ ./q6.sh
11: ./cse141/ders/test1.java
10: ./cse142/test2.java
82: ./q6.sh
0: ./klasor/sun2.txt
0: ./klasor/morning-sun.txt
0: ./klasor/sun1.txt
25: ./klasor/altklasor/dict
0: ./klasor/altklasor/evening-sun.txt
+_____________
Total is 128.
aqui@aqui:~/courses$ ./q6.sh *.java
11: ./cse141/ders/test1.java
10: ./cse142/test2.java
+_____________
Total is 21.
aqui@aqui:~/courses$ ./q6.sh *.java cse141/
11: cse141/ders/test1.java
+_____________
Total is 11.

aqui@aqui:~/courses$ ./q6.sh *.java cse142/
10: cse142/test2.java
+_____________
Total is 10.
aqui@aqui:~/courses$ ./q6.sh *.java cse141/ cse142/
11: cse141/ders/test1.java
10: cse142/test2.java
+_____________
Total is 21.
```