1. Write a bash script that counts all the C files from a given directory and all of its subdirectories.

```
#!/bin/bash

if [ $# -lt 1 ]; then
     echo "Insufficient arguments"
     exit 1
fi

find $1 -type f | grep -E -c "\.c$"
```

2. Write a bash script that counts all the lines of code in the C files from the directory given as command-line argument, excluding lines that are empty or contain only spaces.

```
#!/bin/bash
if [ -z "$1" ]; then
     echo "No parameters given"
     exit 1
fi
if [ ! -d "$1" ]; then
     echo "Parameter is not a folder"
     exit 1
fi
total=0
for f in $(ls "$1" | grep -E "\.c$"); do
  if test -f "$1/$f"; then
     nr_lines=$(grep -E -c -v "^[[:space:]]*$" "$1/$f")
     echo "$f: $nr_lines"
     total=$((total+nr_lines))
fi
done
echo "Total lines: $total"
```

3. Write a bash script that counts all the lines of code in the C files from the directory given as command-line argument and all its subdirectories, excluding lines that are empty or contain only spaces.

```
#!/bin/bash
if [ -z "$1" ]; then
     echo "No parameters given"
     exit 1
fi
if [ ! -d "$1" ]; then
     echo "Parameter is not a folder"
     exit 1
```

```
fi
total=0
for f in $(find "$1" -type f | grep -E "\.c$"); do
     nr_lines=$(grep -E -c -v "^[[:space:]]*$" $f)
     echo "$f - $nr_lines"
     total=$((total+nr_lines))
done
echo "Total lines: $total"
```

4. Write a bash script that receives any number of command line arguments and prints on the screen, for each argument, if it is a file, a directory, a number or something else.

```
#!/bin/bash
while [ ! $# -eq 0 ]; do
     arg=$1
     if test -f $arg; then
          echo "$arg is a regular file"
     elif [ -d $arg ]; then
          echo "$arg is a directory"
     elif echo $arg | grep -E -q "^[0-9]+$"; then
          echo "$arg is an integer number"
     else
          echo "$arg is something else"
     fi
     shift
done
```

5. Write a bash script that keeps reading strings from the keyboard until the name of a readable regular file is given.

```
#!/bin/bash
fname=""
while [ ! -f "$fname" ]; do
     read -p "Enter a string: " fname
done
```

6. Write a bash script that sorts the file given as command line arguments in ascending order according to their file size in bytes.

```
#!/bin/bash
for f in $@; do
     if test -f $f; then
          du -b $f
     fi
done | sort -n
```

7. Write a bash script that calculates the sum of the sizes (in bytes) of all regular files in a folder given as a parameter.(use test to check if the folder exists and if a given file is a regular file)

```
#!/bin/bash

if [ $# -eq 0 ]; then
     echo "Please provide one directory"
     exit 1
fi

if [ ! -d $1 ]; then
     echo "The argument given is not a directory"
     exit 1
fi

sum=0
for f in $(ls $1); do
     if [ -f "$1/$f" ]; then
          size=$(du -b "$1/$f" | awk '{print $1}')
          echo "File: $f - Size: $size"
          sum=$((sum+size))
     fi
done
echo "Total size of regular files from folder $1 is $sum"
```

8. Write a script that reads filenames until the word "stop" is entered. For each filename, check if it is a text file and if it is, print the number of words on the first line.(Hint: test command to check if regular file; file command to check if text file)

```
#!/bin/bash

while true; do
     read -p "Enter filename or stop: " file
     if [ "$file" = "stop" ]; then
          echo "Done"
          exit 0
     elif [ -f "$file" ]; then
          if file $file | grep -E -q "text"; then
               echo "File: $file - Words on first line: $(head -1 $file |
wc -w)"
          fi
     fi
done
```

9. Write a script that receives as command line arguments pairs consisting of a filename and a word. For each pair, check if the given word appears at least 3 times in the file and print a corresponding message.

```
#!/bin/bash

if [ $# -lt 2 ]; then
     echo "Please provide at least 2 arguments"
     exit 1
fi

if [ $(($# % 2)) -eq 1 ]; then
     echo "You must provide an even number of arguments"
     exit 1
fi

while [ $# -gt 1 ]; do
     file=$1
     word=$2

     if [ ! -f "$file" ]; then
          echo "Name $file is not a file"
     else
          count=$(grep -E -o "\<$word\>" "$file" | wc -l)
          if [ $count -ge 3 ]; then
               echo "Word $word appears $count times in file $file"
          fi
     fi
     shift 2
done

if [ $# -eq 1 ]; then
     echo "Warning: final pair is incomplete"
fi
```

10. Write a bash script that sorts all files given as command line arguments descending by size. (first check if an argument is a file)

11. Write a script that extracts from all the C source files given as command line arguments the included libraries and saves them in a file. (use the file command to check if a file is a C source file)

12. Write a script that monitors the state of a given folder and prints a message when something changes.

13. Find recursively in a given directory all the symbolic links, and report those that point to files/directories that no longer exist. Use option -L to test if a path is a symbolic link, and option -e to test if it exists (will return false if the target to which the link points does not exist)

```
#!/bin/bash

for link in $(find "$1" -type l); do
     if [ ! -e "$link" ]; then
     echo "Link $link is not valid"
     fi
done
```

14. Write a bash script that receives a folder name as an argument. Find recursively in the folder the number of times each file name is repeated.

```
#!/bin/bash

if [ -z "$1" ]; then
     echo "Please provide one argument"
     exit 1
fi

if [ ! -d "$1" ]; then
     echo "Argument must be a directory"
     exit 1
fi

find "$1" -type f | awk -F/ '{print $NF}' | sort | uniq -c
```

15. Calculate the average of all process ids in the system per user.

```
#!/bin/bash
# Solution w/o arrays

prev_user=""
count=0
sum=0
for user_pid in $(ps -ef | awk 'NR > 1{print $1","$2}' | sort); do
     curr_user=$(echo "$user_pid" | cut -d, -f1)
     pid=$(echo "$user_pid" | cut -d, -f2)
     if [ "$curr_user" != "$prev_user" ]; then
          if [ $count -gt 0 ]; then
                    echo "Avg for $prev_user is "$((sum/count))
          fi
          prev_user=$curr_user
          sum=0
```

```
          count=0
     fi
     sum=$((sum+pid))
     count=$((count+1))
done
```

16. Write a script that receives program/process names as command line arguments. The script will monitor all the processes in the system, and whenever a program with one of those names is run, the script will kill it and display a message. (see commands ps, kill, killall).

```
#!/bin/bash

if [ $# -eq 0 ]; then
     echo "Provide at least one name"
     exit 1
fi

while true; do
     for process in $@; do
     PIDs=""
     PIDs=$(ps -ef | awk '{print $8" "$2}' | grep -E "\<$process " | awk
'{print $2}')
     if [ -n "$PIDs" ]; then
          kill -9 $PIDs
     fi
     done
     sleep 3
done
```

17. Write a script that receives a directory as a command line argument. The script will delete all the C source files from the directory and will display all other text files sorted alphabetically.

18. Write a script that finds recursively in the current folder and displays all the regular files that have write permissions for everybody (owner, group, other). Then the script removes the write permissions from everybody. Hint: use chmod's symbolic permissions mode (see the manual).

19. Consider a file containing a username on each line. Generate a comma-separated string with email addresses of the users that exist. The email address will be obtained by appending "@scs.ubbcluj.ro" at the end of each username. Make sure the generated string does NOT end in a comma.

```
#!/bin/bash
if [ -z "$1" ]; then
     echo "Please provide one input file"
     exit 1
```

```
fi

if [ ! -f "$1" ]; then
     echo "The given argument is not a file"
     exit 1
fi

result=""
for u in $(cat "$1"); do
     result="$u@scs.ubbcluj.ro,$result"
done

result=$(echo $result | sed -E "s/,$//")

echo $result
```

20. Write a shell script that receives any number of words as command line arguments, and continuously reads from the keyboard one file name at a time. The program ends when all words received as parameters have been found at least once across the given files. Example:

    Assume that
    file1.txt contains word1 and word2
    file2.txt does not contain any of the 3 words
    file3.txt contains word2 and word 3

    ./script.sh word1 word2 word3
    We input the following:
    file1.txt
    file2.txt
    file3.txt

    The program stops after reading file3.txt because
    word1 has been found in file1.txt
    word2 has been found in file1.txt and file3.txt
    word3 has been found in file3.txt

Solution 1:

```
#!/bin/bash

if [ $# -eq 0 ]; then
     echo "Please provide at least one argument"
     exit 1
fi
```

```
declare -A words

for word in $@; do
     words[$word]=0
done

found_all=false
while ! ${found_all}; do
     to_find=""
     for word in ${!words[@]}; do
          if [ ${words[$word]} -eq 0 ]; then
               to_find="$to_find $word"
          fi
     done
     echo "Left to find:$to_find"
     read -p "Please input a filename: " file
     if [ -z "$file" ]; then
          echo "Please input a non empty string"
     elif [ ! -f "$file" ]; then
          echo "$file is not a file"
     else
     found_all=true
     for word in $@; do
          if grep -E -q "\<$word\>" "$file"; then
               echo "Found $word in $file"
               words[$word]=1
          fi
          if [ ${words[$word]} -eq 0 ]; then
               found_all=false
          fi
     done
     fi
done

echo "All done"
```

Solution 2:

```
#!/bin/bash

if [ $# -eq 0 ]; then
     echo "Please provide at least one argument"
     exit 1
fi
```

```
found_all=false
all_files=""
while ! ${found_all}; do
     read -p "Please input a filename: " file
     if [ -z "$file" ]; then
          echo "Please input a non empty string"
     elif [ ! -f "$file" ]; then
          echo "$file is not a file"
     else
     all_files="$all_files $file"
     found_all=true
     for word in $@; do
          if grep -E -q "\<$word\>" $all_files; then
               echo "Found word $word"
          else
               found_all=false
          fi
     done
     fi
done

echo "All done"
```

21. Write a shell script that, for all the users in /etc/passwd, creates a file with the same name as the username and writes in it all the ip addresses from which that user has logged in. (hint: use the last command to find the ip addresses)

```
#!/bin/bash

destination="./results"
if [ ! -d $destination ]; then
  if [ ! -e $destination ]; then
     mkdir $destination
  else
     echo "The file $destination already exists and it is not a directory.
Exiting."
     exit 1
  fi
fi

users=$(awk -F: '{print $1}' /etc/passwd)

for user in $users; do
  ips=$(last $user | head -n -2 | awk '{print $3}' | sort | uniq)
  if test -n "$ips"; then
     echo "$ips" > $destination/$user
```

```
  fi
done
```

22. Create a bash script that displays every second the process count per user sorted descending by process count for all users specified as command line arguments. If no arguments are given, the script will display the process count per user for all users.

```
#!/bin/bash

users="-e"
if [ $# -gt 0 ]; then
     users=""
     for user in $@; do
     users="$users -u $user"
     done
fi

while true; do
     clear
     ps -f $users | awk 'NR > 1{print $1}' | sort | uniq -c | sort -n -r
-k1,1
     sleep 1
done
```

23. Create a bash script that finds all the text files in a specified folder (the current folder if there is no specified folder). For all such files, the script will report the filesize, permissions, and number of unique lines.

```
#!/bin/bash

dir=${1:-"."}

if [ -d "$dir" ]; then
     for f in $(find "$dir" -type f); do
     if file $f | grep -E -q "text"; then
          size=$(du -b $f | cut -f1)
          perm=$(ls -l $f | cut -d' ' -f1)
          lines=$(sort $f | uniq | wc -l)
          echo "Filename: $f - size: $size - permissions: $perm - unique
lines: $lines"
     fi
     done
fi
```

24. Write a bash script that receives as command line arguments pairs of arguments A and B. For each pair, if argument A contains argument B, display a message.

```
#!/bin/bash
while [ $# -ge 2 ]; do
     a=$1
     b=$2
     if echo $a | grep -E -q $b; then
          echo "String $b is found in string $a"
     fi
     shift 2
done

if test $# -gt 0; then
     echo "Incomplete pair: $1 - $2"
fi
```

25. Write a bash script that receives as command line arguments the names of either files or directories.
    - If the argument is a regular file, then display the first 10 lines from that file.
    - If the argument is a directory, create a file in that directory with the same name as the directory + ".info", and store the output of the ls -l in that file.
    - If the argument is neither a regular file or a directory, display a message.

```
while test $# -ge 1; do
     if test -f $1; then
          # head -n 10 $1
          awk 'NR < 11 {print $0}' $1
     elif test -d $1; then
          # get the basename of the directory,
          # in case we have an entire path: /dir1/dir2/dir3 -> the
     basename of the directory we are working with is dir3
          # d_name=`basename $1`
          d_name=`echo "$1" | awk -F/ '{print $NF}'`
          f_name="${1}/${d_name}.info"
          ls -l $1 > ${f_name}
     else
          echo "The argument $1 is neither a regular file nor a directory"
     fi
     shift
done
```

26. Write a bash script that receives any number of command-line arguments. For each argument that is a directory, find in it recursively and display:
    - all the subdirectories that are empty or contain only hidden files.
    - all the files that are empty or contain only whitespaces (in their content, not in their name).

    Note: hidden files always have a name that starts with "."

```
#!/bin/bash
for x in $@; do
     if [ -d $x ]; then
     for y in $(find $x); do
          if [ -d $y ]; then
               # option 1:
               # ls, by default, omits entries beginning with .
               dir_content=$(ls $y | head -1)
               if [ -z "$dir_content" ]; then
                    echo "Empty dir: $y"
               fi
               # option 2:
               # the first grep removes the directory itself from the
printed values and the second grep looks for at least one entry that does
not begin with .
               # if ! find $y | grep -E "$y/" | grep -E -v -q "^$y/\.";
then
               #    echo "Empty dir: $y"
               # fi
          elif [ -f $y ]; then
               # option 1 (the very weird way of doing it):
               # replace all whitespaces with nothing, then filter only
the lines that still contain at least one character, then count the
caracters; if count > 0, the file must contain something other than
whitespaces
               # character_count=$(sed -E "s/\s//g" $y | grep -E "." | wc
-c)
               # if [ $character_count -eq 0 ]; then
               #    echo "Empty file: $y"
               # fi
               # option 2:
               if ! grep -E -q -v "^\s*$" $y; then
                    echo "Empty file: $y"
               fi
               # there are more ways to do the same thing, but 2 very
different approaches are enough for now
          fi
     done
     else
```

```
          echo "Whatever you gave me here: $1, is not a directory and,
sincerely, you should be ashamed of yourself."
     fi
done
```

27. Write a bash script that receives any number of command-line arguments. For each argument that is a directory, find recursively in it and display:
    - all the subdirectories that contain at least one non-hidden file
    - all the files that contain at least 10 non-empty lines

    Note: hidden files always have a name that starts with "."

```
#!/bin/bash
for x in $@; do
     if [ -d $x ]; then
     for y in $(find $x); do
          if [ -d $y ]; then
               # option 1:
               dir_content=$(ls $y | head -1)
               if [ -n "$dir_content" ]; then
                    echo "Non empty dir: $y"
               fi
               # option 2:
               # the first grep removes the directory itself from the
printed values and the second grep looks for at least one entry that does
not begin with .
               # if  find $y | grep -E "$y/" | grep -E -v -q "^$y/\.";
then
               #    echo "Non empty dir: $y"
               # fi
          elif [ -f $y ]; then
               count=`grep -E -c "." $y`
               if [ $count -gt 10 ]; then
                    echo "File with more than 10 non empty lines: $y"
               fi
          fi
     done
     else
          echo "Why is this thing: $1, a not-directory, in my arguments?"
     fi
done
```