Q1. Write a bash script to print the directory tree (iterate over sub-directories as well) from a directory given as a parameter to the script. While printing the name of the files, remove the extension of the file. Also, count the number of files and directories.

The file **mytree.sh** is as follows.

```
#!/bin/bash

BRANCH_SYMBOL="└──"
DIR_COUNT=0
FILE_COUNT=0

# returns a string that represents the name of the file without extension
# $1: file name
remove_extension() {
    FILE_NAME=$(echo $1)
    REV_FILE_NAME=$(echo $1 | rev)

    for (( i=0; i<${#REV_FILE_NAME}; i++ ))
    do
        if [ "${REV_FILE_NAME:$i:1}" = "." ]
            then
                INDEX=$((i + 1))
                FILE_NAME=$(echo "${REV_FILE_NAME:$INDEX:${#FILE_NAME}}" | rev)
                break
        fi
    done

    echo $FILE_NAME
}

# prints the name of the directory or file
# $1: number of offset
# $2: name of file(not full path)
# $3 : 1 if file otherwise 0
print() {
    SPACE_LEN=$(( $1 * 4 ))
    EMPTY_STRING=$(printf %${SPACE_LEN}s)
    LINE="${EMPTY_STRING}${BRANCH_SYMBOL}"
    FILE_NAME="$2"
    if [ $3 -eq 1 ]
        then
            FILE_NAME=$(remove_extension $FILE_NAME)
    fi
    echo "${LINE}${FILE_NAME}"
}

# recursively traverses the directory and prints the name of directory and file.
also calculates the total number
# of directories and files in global variables defined at the beginning of file
# $1: path to directory
# $2: number of offset
dfs() {
```

```
    for i in $(ls $1)
    do
        if [ -d "${1}/${i}" ]
            then
                (( DIR_COUNT++ ))
                print $2 "${i}" 0
                dfs "${1}/${i}" $(( $2 + 1 ))
            else
                (( FILE_COUNT++ ))
                print $2 $i 1
        fi
    done
}

# call the dfs function and pass the name of the directory given as arg. second
argument represents the indentation level
dfs $1 0

# print the count of directories and files
echo
echo "${DIR_COUNT} directories, ${FILE_COUNT} files"
```

Following command is used to run the program.

```
./mytree.sh ../../../assignments
```

The output of the above script is as follows.

```
        └──input
        └──main
        └──matrix
        └──matrix
        └──screenshot
            └──1
    └──4
        └──first
        └──first
        └──main
        └──main
        └──main-static
        └──mylib
        └──mylib
        └──second
        └──second
    └──MIT2021117-PPR-A2
    └──PPR-Assignment-2
└──3
    └──1
        └──1-1
        └──1-2
        └──1-3
        └──1-4
```

```
        └──1-2
        └──1-3
        └──1-4
    └──2
        └──2-1
        └──2-2
        └──2-3
        └──2-4
        └──2-5
        └──2-6
        └──cpu-bound
        └──cpu-bound
        └──io-bound
        └──io-bound
    └──3
        └──a
        └──notes
        └──output
        └──q3
    └──4
        └──4-1
        └──4-2
        └──a
        └──q4
```

```
        └──4-2
        └──a
        └──q4
    └──5
        └──a
        └──output
        └──q5
    └──6
        └──a
        └──management
        └──q6
    └──7
        └──a
        └──client
        └──output
        └──server
        └──server
    └──8
        └──inputs
            └──input
        └──input
        └──output-1
        └──output-2
        └──output-3
```


└── sort1
└── sort1
└── xsort
└── xsort
└── 9
└── notes
└── row-mul
└── row-mul
└── row-mul
└── serial-mul
└── serial-mul
└── serial-mul
└── notes
└── PPR-assignment-3
└── questions
└── 5
└── 1
└── mytree
└── 2
└── assig5-shell
└── PPR-assignment-5.md
23 directories, 83 files
@1 👀

Q2. Write a bash script to implement bubble sort.

The required script is as follows.

bubble.sh file:

```
#!/bin/bash

arr=($(cat $1))
N="${#arr[@]}"

echo "The original order of array is as follows."
echo ${arr[*]}

# Performing Bubble sort
for (( i=0; i<N-1; i++ ))
do
    for (( j=0; j<N-i-1; j++ ))
    do
        if [ ${arr[$j]} -gt ${arr[$(( j+1 ))]} ]; then
            temp=${arr[j]}
            arr[$j]=${arr[$(( j+1 ))]}
            arr[$(( j+1 ))]=$temp
        fi
    done
done

echo "After sorting, array is as follows."
echo ${arr[*]}

# write the sorted array back to file
echo ${arr[*]} > $1
```

Following command is used to run the program.

```
./bubble.sh input.txt
```

The output of the above script is as follows.

```
@2 👀 ./bubble.sh input.txt
The original order of array is as follows.
898 787 79 798 2 4 3 332 24 1 0 89 32 3
After sorting, array is as follows.
0 1 2 3 3 4 24 32 79 89 332 787 798 898
@2 👀
```

Q3. Write a bash script that should compile a .c program, run the program and then save the output of the program in an output file.

The required script is as follows.

run.sh file:

```
#!/bin/bash

gcc $1 -o "$1.out"
touch "$1.output.txt"
"./$1.out" > "$1.output.txt"
```

The sample C program to demonstrate the usage of this script is as follows.

hello.c file:

```
#include<stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

Following command is used to run the program.

```
./run.sh hello.c
```

Its output is as follows.

Q4. Write a program to flip a coin, it should show some animation of ASCII characters beforeprinting the result of the coin flip and also give a beep while showing the result. To give a beep,you need to first install beep package in your system. Use the sample output as shown below.

The required script is as follows.

coin.sh file:

```
#!/bin/bash

# head face
HEAD="
 _____
/HEADS\\
\\_____/
\\|||||/
"

# tail face
TAIL="
 _____
/TAILS\\
\\_____/
\\|||||/
"

# toss the coin
DURATION=3
while [ $DURATION -ne 0 ]
do
    echo -e "$HEAD"
    echo "Tossing the coin..."
    sleep 1
    clear
    echo -e "$TAIL"
    echo "Tossing the coin..."
    sleep 1
    clear

    (( DURATION-- ))
done

# use $RANDOM to get result of toss
RESULT=$(( 1 + $RANDOM % 2 ))

# play the beep sound
beep
# print the heads/tails depending on result
echo "Result of toss is..."
if [ $RESULT -eq 1 ]; then
    echo -e "$HEAD"
    else
        echo -e "$TAIL"
fi
```

To play the "beep" sound, we need to install the beep package. I've used following command in my fedora machine to install the package.

```
sudo dnf install beep
```

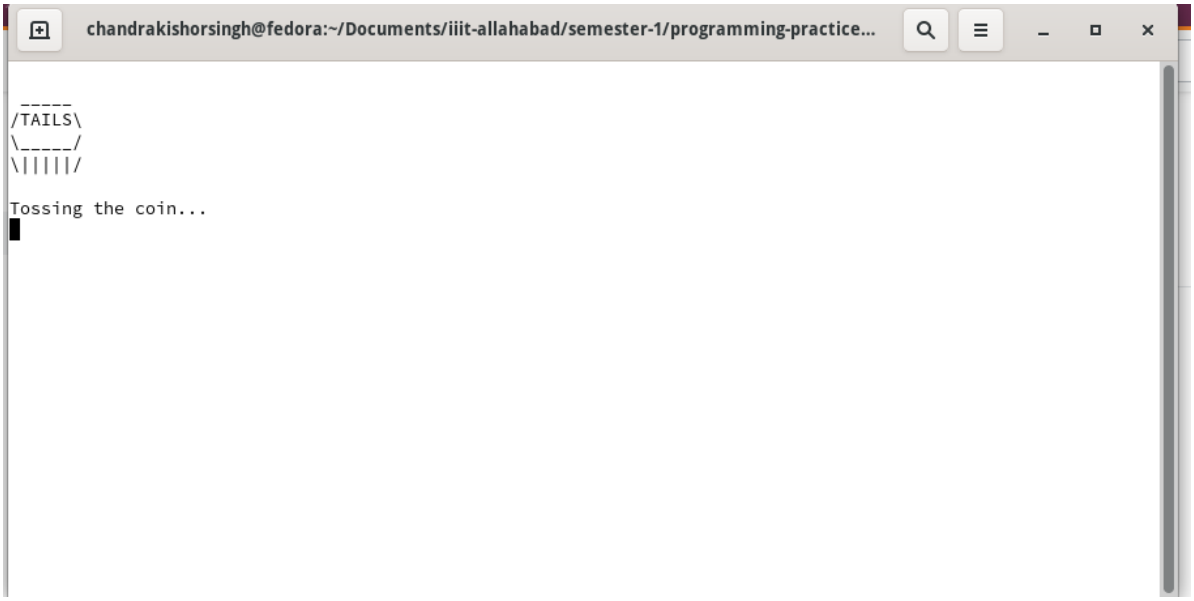
Also, the `pcspkr` kernel module should be loaded for `beep` package to work. Following command is used to load the required kernel module.

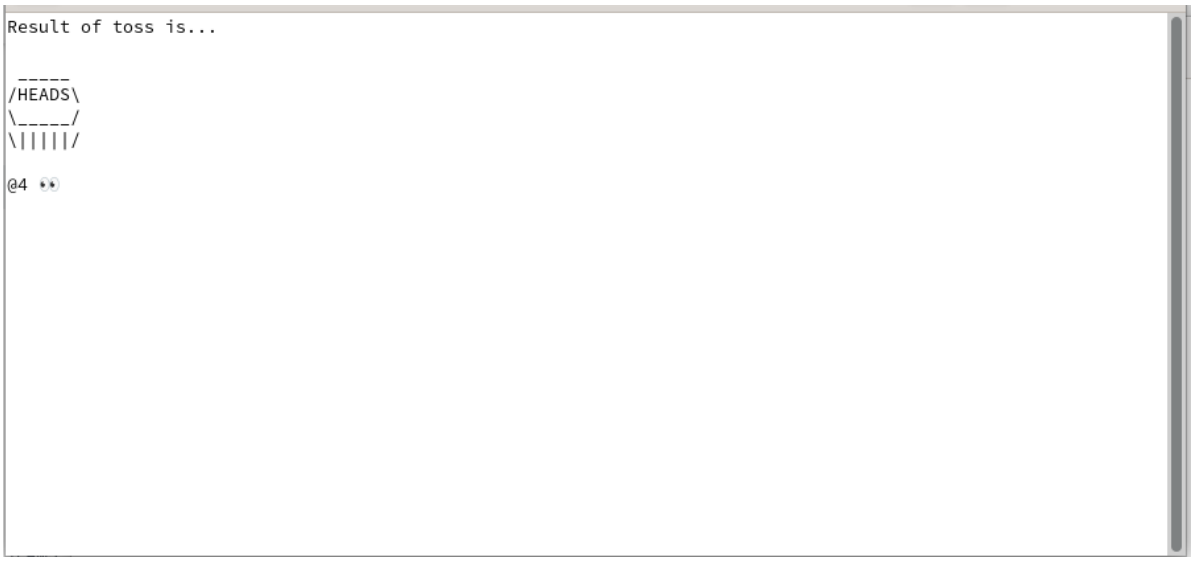
```
sudo modprobe pcspkr
```

Following command is used to run the program.

```
./coin.sh
```

Its output is as follows.





Q5. Explain the usage of the following bash script.

The given script is as follows.

```
#!/bin/bash
limit=5 # seconds to wait for timeout
file=$1
timeout=$(awk "BEGIN{srand(); print srand() + $limit}")
echo $timeout
until [ -s "$file" ] || [ $(awk 'BEGIN{srand();print srand()}') -gt $timeout ]
do
sleep 1
done
if ! [ -s "$file" ]
then
# timed-out
exit 1
else
cat $file
fi
```

The given script will print the current time + 5 and the content of the file(provided as first argument to the script) if the given path represents a valid file and is not empty. Otherwise, it will just print the timeout and wait for 5 seconds to give back the prompt.

I have commented the given code to explain the main logic of the program. The commented code is as follows.

```
#!/bin/bash

# assign `5` to `limit` variable
limit=5 # seconds to wait for timeout
# assign the first argument to `file` variable
file=$1

# initialize the srand() function by current time and assign a `current time + 5`
to the `timeout` variable
timeout=$(awk "BEGIN{srand(); print srand() + $limit}")
# print the `timeout` variable
echo $timeout

# until the current time is not greater than `timeout` variable or the file(whose
path is assigned to `file` variable) is exists and is not empty, sleep for 1
second
until [ -s "$file" ] || [ $(awk "BEGIN{srand();print srand()}") -gt $timeout ]
do
    sleep 1
done
```

```
# if file does not exists or if it is empty, then exit with status of 1.
otherwise show the contents of file using cat command.
if ! [ -s "$file" ]
then
    # timed-out
    exit 1
else
    cat $file
fi
```

Its output is as follows.