```
#!/bin/bash

# print the current working directory you are in

mypwd()
{
      pwd
}

# print only the files in the current working directory
# files should not include hidden files.

mydir()
{
      ls
}

# print  all the files - including hidden files.
# in the current working directory

mydirh()
{
      ls -a
}

# print all files - including hidden files attributes in
# the current working directory.

mydira()
{
      ls -al
}

# print all the attributes as above but in a human
# human readable format.

mydirhuman()
{
      ls -alh
}

# print all the attributes of files starting with
# the alphabet d and ending with y.

mydirgen()
{
      ls -alh d*y
}

# create a regular file  name with zero size
# the file name should be cafe.c

mycreate()
{
      touch cafe.c
}

# create a directory file name cafedir
#

mydircreate()
{
      mkdir cafedir
}
```

```
# remove the file cafe.c you created above.

myremove()
{
      rm cafe.c
}

# remove the dir cafedir you created above.

myremovedir()
{
      rmdir cafedir
}

# create a directory called as cafedir, and create
# ten files of the form cafe1, cafe2..
# up to cafe10.

myfiles()
{
      mkdir cafedir
      cd cafedir
      for i in {1..10}
      do
            touch cafe$i
      done
      cd -
}

# remove the directory cafedir completely.

myremove_cafedir()
{
      rm -rf cafedir
}

# print the path of the command passed in
# agument 1. eg, mycmd_path ls should print
# /usr/bin/ls.
#

mycmd_path()
{
      which $1
}

# you have a file of "n" lines.
# this command will arbitrarily print the
# lines ranging from n1 to n2, passed in
# arg2 and arg3 of this function. arg1
# will have the file name.
# argument 1-  file name
# argument 2 - n1
# argument 3 - n2

myfrange()
{
      (( d=$3-$2+1 ))
      head -n $3 $1 | tail -n $d
}

# print the total no. of lines in the file
# argument 1 - is the file name.
```

```
myflines()
{
      x=`wc -l $1 | awk '{print $1}'`
      echo $x
}

# create an alias "myg" for doing the following
# routine. Compile a file mysort.c (assume this file
# will be available) to create an executable "mysort",
# execute the executable with an input file input1,
# and store the output in an output file output1.
# when I type "myg" it will do all of this.

myalias()
{
      alias myg='gcc -o mysort mysort.c; ./mysort < input1 > output1'
}

# The following function will remove the
# first "n" arguments from the passed arguments
# after the first argument (first argument is the
# value of "n") see eg. below
# (leaving arg0 as is ), and print only the remaining
# arguments along without arg0 in the beginning
# and next line it will print the total no. of
# arguments remaining after removing the "n"
# plus the first argument that carried the value of n.

# argument1: "n" , no. of arguments to be removed.
# argument 2 ... on wards any arbitrary arguments
# and any number of them. "n" arguments from this
# list to be removed plus the argument 1 as well.

# eg., myargs 3 1 2 3 4 5 6 7
# gives the following output:
# 4 5 6 7
# 4
# the second line above has to be printed the right way
# using shell no. of arguments variable.

myargs()
{
      a=$1
      shift 1
      shift $a
      echo "$@"
      echo $#
}

# print sum of two integers passed here to stdoutput
# arg1 - first integer
# arg2 - second integer

mysum()
{
      (( sum=$1+$2 ))
      echo $sum
}

# print the no. of files in the current dir
#

mydirfiles()
```

```
{
      x=`ls | wc -l`
      echo $x
}

# return 0 if the two files are same
# else return 1
#dont print any thing in stdoutput
# Argument 1 : file1
# argument 2 : file 2

mydiff()
{
      diff  $1 $2
      if [[ $? == 0 ]]
      then
            return 0
      else
            return 1
      fi
}

# append two characters a and b to
# the file passed in argument 1

myappend()
{
      echo "ab" >> $1
}

# in this function we will pass the arguments of
# many filenames. function should concatenate all
# file contents to the first file argument.
# if f1, f2, ... fn are arguments
# all file data from f2 till fn should be concatenated
# to f1.

myconcatenate()
{
      f1=$1
      shift 1
      for i in $@
      do
            cat $i >> $f1
      done
}

#
# I will pass a list of file paths, some of which
# could be directories as well. This function will do
# a ls -l of each of the path, and print the successful
# outputs in the terminal. And the errors should not
# be printed in the output, as some of the file paths
# may not even exist.
#
# Arguments: any no. of arguments of file paths.
# it could be 1 file , 2 files, 10 files or 100 files.

myattrs()
{
      for i in  $@
      do
            ls -l $i 2>/dev/null
      done
```

```
}

# the following function is same as above, except
# that all good output will go to a file
# "attr.out", and all errors will go to a file
# "attr.errs".
# Arguments: any no. of arguments of file paths.
# it could be 1 file , 2 files, 10 files or 100 files.
# function should not print anything.

myfattrs()
{
      for i in  $@
      do
            ls -l $i >> attr.out 2>>attr.errs
      done
}

# As in this file you have my exam file template
# where all functions start with "my..."
# given this file as an argument, this function
# should calculate how many such functions
# are there.
# No other line in the file starts with my except
# these function names, however all the other
# parts of the file can have the word "my"
# embedded in it.
#
# Argument 1: file name
# output: no. of functions given the format as
# discussed above.

myfunctions()
{
      grep ^my $1 | wc -l
}


# create the directory hierarchy as follows:
# from the current working directory.
# d1/d2/d3
# d1/d2/d4
# d1/d5/d6
# d1/d5/d7
# copy the file say original file name of (sent as argument 1)
# to all the above directories,
# in the above there should be total of 7 files each one
# in each of directories d1 through d7.
# and then return 0.
# Note, you should create the directory hierarchy from d1
# through d7 and then copy the files.

# Argument 1 - filename that needs to be copied to all dirs.

mycopy_files()
{
      mkdir -p d1/d2/d3
      mkdir -p d1/d2/d4
      mkdir -p d1/d5/d6
      mkdir -p d1/d5/d7
      cp $1 d1/d2/d3
      cp $1 d1/d2
      cp $1 d1
      cp $1 d1/d2/d4
```

```
	cp $1 d1/d5/d6
	cp $1 d1/d5/d7
	cp $1 d1/d5

}

# You have file that has an integer value
# in each line. ie. line no. 1 is 1, line no. 2 is 2
# and so on. line n will have the number "n".

# below you will implement mysumrange() where you will
# now sum the range n1 - n2 of the file passed
# in argument 1. If n1 greater than the no. of
# lines in the file sum will print 0.
# if n2 is greater than the no. of lines in the file
# sum will print sum between n1 and the last line of the file.
# if n1 == n2 print the value in that line as sum.

# Argument1   - is the file name.
# Argument 2 - starting line number n1
# argument 3 - ending line number n2.
# output - sum of all numbers in those lines in the file.

# Hint: Sum of consecutive positive numbers from a to b

myfsum()
{
	nlines=`wc -l $1  | awk '{print $1}'`
#	echo "nlines: $nlines"
	if (( $2 > $nlines ))
	then
		echo 0
		return
	elif (( $3 > $nlines ))
	then
		(( d=$nlines-$2+1 ))
		(( sum=d*($nlines+$2)/2 ))
		echo $sum
		return
	else
		(( d=$3-$2+1 ))
		(( sum=d*($3+$2)/2 ))
		echo $sum
		return
	fi

}

# in the below function
# you will print the sizes of each file in the
# current directory. Only for regular files and not for
# directories.
# <filename>:<size>
# should be the format of the output
# size should be in bytes.

# hint: [[ -f <filename> ]] will return true for
# regular files.

myfsizes()
{
	for i in *
	do
		if [[ -f $i ]]
```

```
            then
                  size=`ls -l $i | awk '{print $5}'`
                  echo "$i:$size"
            fi
      done

}

# given a list of files with full path names in a file passed in
# argument 1 this function will print the directory names
# where the files reside.
# eg., if the file path is /d1/d2/d3/f1 print the
# directory as /d1/d2/d3. The output should be:
# f1:/d1/d2/d3
# f2:/d1/d2/d4.. etc.
# <filename>:<directoryname>

# Argument 1 : filename that has the list of file paths.
# each separated by a new line. (ie. one line for every filepath).
# Do not print any extra space/characters/new lines.
# a list of contiguous lines in the above format.
# output: print the above output as mentioned.

mynames()
{
      for i in `cat $1`
      do
            echo `basename $i`:`dirname $i`
      done
}

# THe command find is a useful utility to print all
# files underneath a current working directory.
# for eg.,
# find . -type f -print
# will print all the file names relative to the
# current working directory that reside below it.
# if the dir structure is d1/d2/d3
# and d1 has f1, d2 has f2, and d3 has f3
# find . -type f -print
# will print
# d1/f1
# d1/d2/f2
# d1/d2/d3/f3
#
# Note: it wont print directories owing to the "f"
# type passed for only files.
#
#
# given the above command, the function below
# will print the total lines of all the regular files
# in the current working directory.
#
# eg. in the above eg. if f1,f2, and f3,  has 10 lines, your function
# will print 30 for the above example.

# use for loop with find command

mytlines()
{
      sum=0
      for i in `find . -type f -print`
      do
            x=`wc -l $i | awk '{print $1}'`
```

```
            (( sum=x+sum ))
      done
      echo $sum
}


# You have two files f1 and f2
# sent as arguments to this function.
# the function will print if the file contents
# are same between line no. n1 and n2 in the
# two files. Assume n1 and n2 will both be
# less than the total lines of f1 and f2.
#
# Arg1 - filename 1
# Arg2 - filename 2
# Arg3 - n1
# Arg 4 - n2
#
# return 0 if the contents are same.
# return 1 if the contents are different.

# Hint: look at the myfrange function you implemented.

myrangediff()
{
      f1=$1
      f2=$2
      n1=$3
      n2=$4

      echo "mydiffrange: $@"
      rm -f f1.out f2.out
      (( d=$n2-$n1+1 ))
      echo $d
      head -n $n2 $f1 | tail -n $d   > f1.out
      head -n $n2 $f2 | tail -n $d  > f2.out
      diff f1.out f2.out
      if [[ $? == 0 ]]
      then
            return 0
      else
            return 1
      fi
}

# You have a file with comma separated fields
# each line will have the following format.
# line1 : a,b,c,d,e
# line2:  cat,dog,snake,father
#....

# The function below will take that file name as argument,
# and print out only the fields I request.

# Argument 1: file name
# Argument2 : field 1
# argument 3: field 2
# .. etc.,
# assume atleast one argument for fields.

# Hints:
# cut  - utility.
# cut -d , -f1,2 will print the fields 1 and 2 with
# delimiter "," as in csv file.
```

```
myfields()
{
      fname=$1
      shift 1
      fields=$1
      shift 1
      for i in $@
      do
            fields="$fields",$i
      done
      cut -d , -f$fields $fname
}
```