CMPE 230 ASSIGNMENT 2 REPORT

Elif Çalışkan 2016400183
Yağmur Kahyaoğlu 2015400057

HOW TO COMPILE:

In a terminal, call commands:

> python filelist.py [options] [directory list]

For example :

python filelist.py –smaller 1K –duplname –zip 'abc.zip' /home/yagmur/Desktop/folder1 /home/yagmur/Desktop/folder2

You should run the program in Linux.

1)PROBLEM DESCRIPTION

In this project we are asked to implement a file utility program called filelist that will traverse directories and report path names of files that satisfy some search criteria such as bigger, smaller, match, before, after, zip, delete, duplcont, duplname, stats and nofilelist.

-**before datetime** Files last modified before a date (YYYYMMDD) or a datetime (YYYMMDDTHHMMSS)
-after datetime Files last modified after a date (YYYYMMDD) or a datetime (YYYMMDDTHHMMSS)
-match Filenames matching a Python regular expression
-bigger Files having sizes greater than or equal to bytes. Note that can also be given as kilobytes, megabytes and gigabytes, for example as, 2K, 3M, 7G.
-**smaller** Files having sizes less than or equal to bytes. Note that can also be given as kilobytes, megabytes and gigabytes, for example as, 2K, 3M, 7G.
-**delete** The files should be deleted.
-zip The files should be packed as zipfile.
-duplcont Files whose contents are the same should be listed. The listing should be in sorted order with a line of ------ characters separating the duplicate sets of files.
–**duplname** Files whose names are the same should be listed. The listing should be in sorted order with a line of ------ characters separating the duplicate sets of files.
-**stats** Traversal statistics should be printed at the end of files listing output. The statistics will be as follows: total number of files visited, total size of files visited in bytes, total number of files listed, total size of files listed in bytes. If –duplcont option is given, additionally, the following information will be printed: total number of unique files listed, total size of unique files in bytes. If –duplname option is given, additionally, the following information will be printed: total number of files with unique names.
-**nofilelist** No file listing will be printed

2)PROBLEM SOLUTION

The arguments in [ ] are optional. If they are not given, the default action is carried out. If no directory list is given, the default is the current directory. If no option arguments are

given, pathnames of all files are printed by the program. If multiple options are present, the files satisfying conditions of all options are returned.

We have three queues: oplist for options, qlist for paths and printlist for the items that will be printed.

First while loop takes every argument and puts them into different queues according to their types. If the argument is an option with another string then it combines the two in s string s and appends it to oplist, else if it is an option with a single string then it directly appends it to oplist. Else, if the argument is a path that exists, we append it to qlist. If it doesn't exist we print a message.

Second while loop checks every option in the oplist and increments the variables with their names to count the number of every option. If option is **before** or **after** it stores the date information that comes with the option in adate or bdate. If it is match, then it stores the expression in exp. If the option is **bigger** or **smaller**, it takes the size information and converts it into bytes. If the option is **zip** it takes the file name and stores it in zipf.

If any of the options are called more than once it means there is an error and the variable error becomes 1. Else if duplname or duplcont is called with delete then it also means an error. The next two elif blocks controls if there is an error in the format of the dates given with before and after.

We start tracing the files if there is no error in options. If there is no path given in arguments so that the qlist is empty, then we add the current directory to qlist else while there is an element in qlist the loop first finds the items in given directories and adds them to qlist. Then one by one it pops them from the queue to analyze the files. For every file we check the **before**, **after**, **match**, **bigger** and **smaller** options appearing with different combinations. We check currentitem's modification date, size and name accordingly and append it to printlist if it passes the tests.

For **zip** option we imported zipfile and we opened the zipfile with 'w' mode. Since there cannot be more than one file with the same name, we check if there exists a file with the same name and we look for the number that the filename appears. Then we add the number minus one with the parentheses around it. After updating the name, we write that file to the zip.

If **duplcont** option is called, it means we should list the files with the same content. First we open and read every file in printlist, content of every file is stored in data variable. if data is not in duplc list, we append data to duplc. Then we append y to its index in matrixc nested list. matrixc is a nested list and it is used for keeping the files with same content at the same index. Then we sort every list in matrixc according to their filename, after bubblesort, if nofilelist is not 1 and the length of the list is greater than 1, we print every list with same content between the dashes. This list is sorted alphabetically according to their filenames then we print other files under these files. They are not sorted, they are listed according to their traversal sequence. For example if the filenames with the same content are [A, C] and the filenames to be listed are [C, B, A, E, D],the output without dashes will be [A, C, B, E, D]. A and C are sorted according to their file names, B, E, D are listed according to the traversal.

If **duplname** option is called, it means we should list the files with the same name. First we find the part that starts with the last '/' in a while loop. If this name is not in list dupl, we append the name to dupl. Then we append y to its index in matrixn nested list. matrixn is a

nested list and it is used for keeping the files with same name at the same index. Then we sort every list in matrixn, after sorting, if nofilelist is not 1 and length of every list is greater than, we print every list with same name between dashes. Listed files are listed alphabetically according to their path. Then we print other files under these files. They are not sorted, they are listed according to their traversal sequence. For example if the filenames to be listed are [B, A, F, b/D, a/D, d/E, e/E, C], the output without dashes will be [a/D, b/D, c/E, d/E, B, A, F, C]. b/D, a/D, and d/E, e/E are sorted according to their paths, A ,B, C, F are listed according to the traversal.

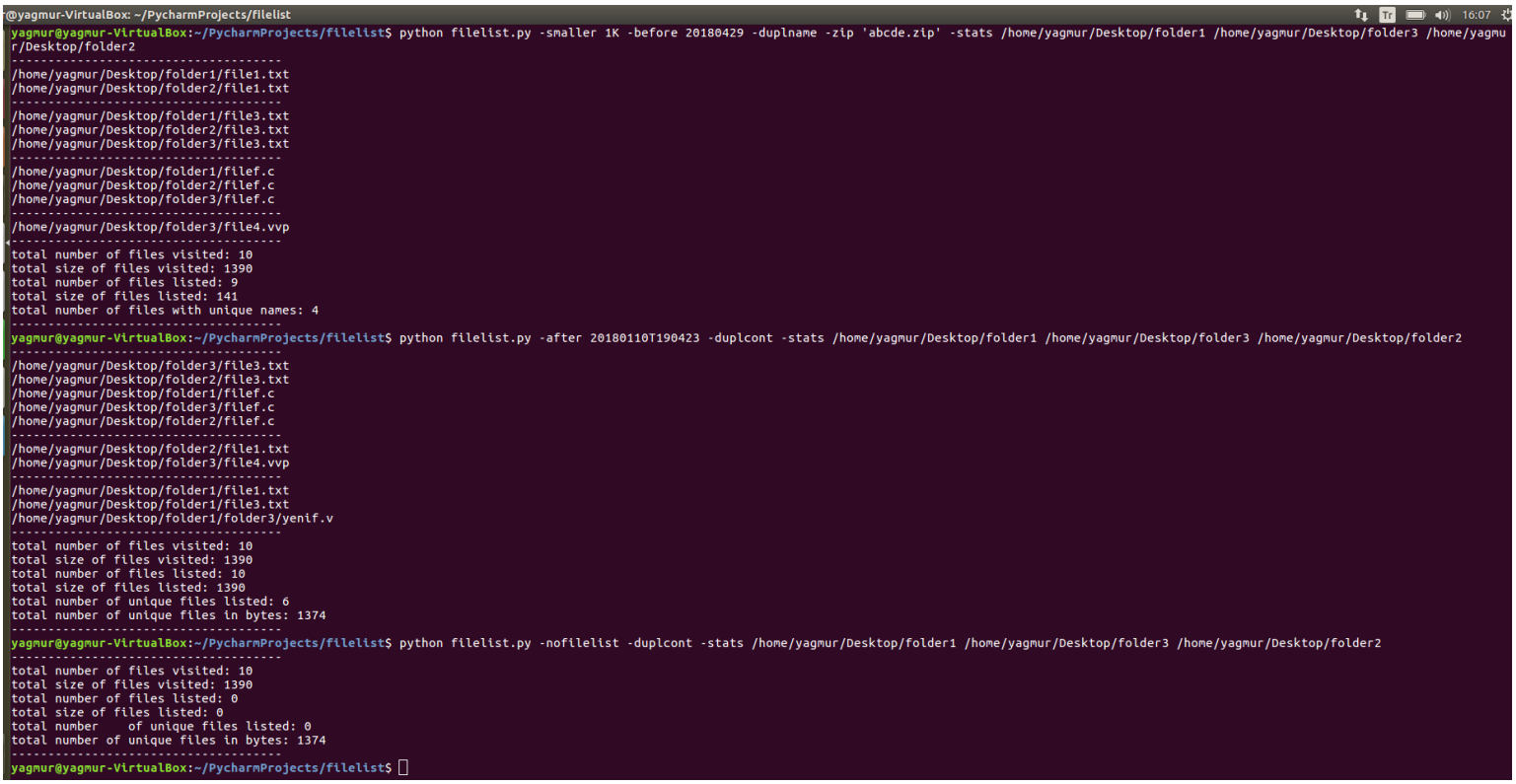
If there exists a **delete** option then we take every file from printlist(if the list is not empty) and add them to a string called ccommand with the linux command 'rm' at the beginning. Then by using os.system we execute the command.

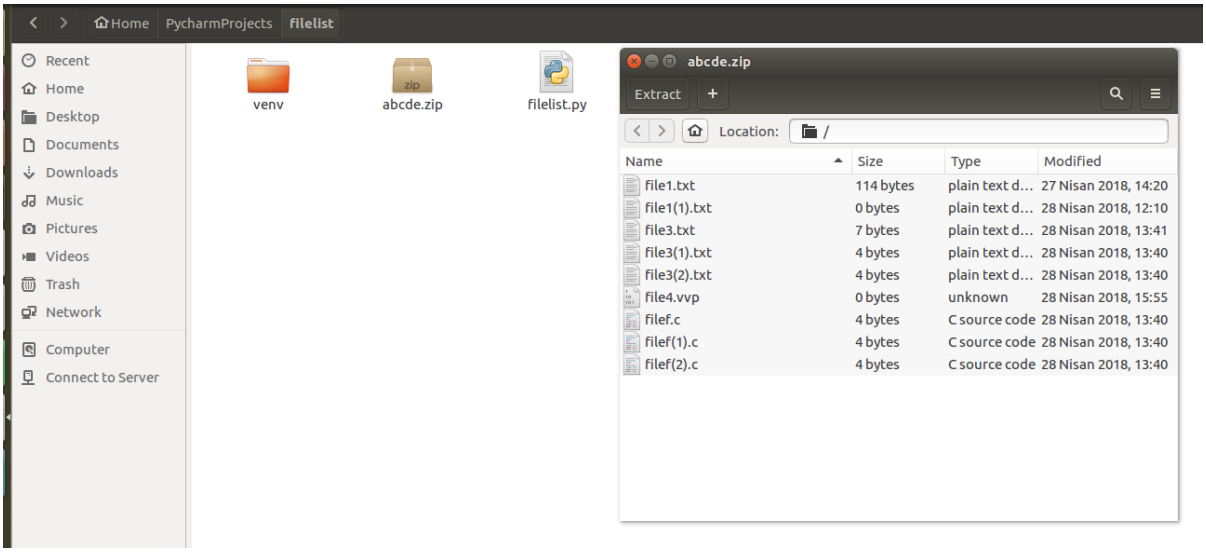
If **nofilelist**, **duplcont** nor **duplname** is called we simply print the names of the files that are in printlist.

If **stats** option is called, it means we should print the files at the end of files listing output. First, we iterate over every string in printlist, we find its size and we add it to lbytesize to find total size of listed files, we find the length of printlist and this number is total number of listed files. We declared vnumFiles and vbyteFiles earlier and when we traverse the directory we incremented their values. If stats and duplcont are called together, we should also write the number of unique files and total size of unique files. We used matrixc for finding unique files if stats and dupl are called together, we should also write the number of files with unique names. We used matrixn for finding files with unique names.

If error is not 0 we print an error message and terminate.

3)EXAMPLES





4)CONCLUSION

To sum up, we implemented a file utility program called filelist that will traverse directories and report path names of files that satisfy some search criteria. We took the system arguments which consist of options and directories. We traversed every file in given directories and implemented the given options. If there wasn't any directory listed, we used our current directory. We considered every combination of the options and tested every file according to them. At the end we printed list of files that passed every test. If there wasn't any options given, we printed all the files in given directories.

4)SOURCE CODE

```
#!/usr/bin/env python

from __future__ import print_function
import os
import sys
import pwd
import subprocess
import datetime
import re
from collections import deque
import zipfile

error = 0

before = 0
after = 0
match = 0
bigger = 0
smaller = 0
delete = 0
zip = 0
duplcont = 0
```

```
duplname = 0
stats = 0
nfl = 0

adates = ''
bdates = ''
bnewsize = 0
snewsize = 0

n = len(sys.argv)

oplist = deque('t')         # list that stores options
test1 = oplist.popleft()
qlist = deque('t')          # list that stores paths
test2 = qlist.popleft()
printlist = deque('t')      # list that stores the items that will be printed
test3 = printlist.popleft()

vnumFiles = 0
vbyteFiles = 0
lnumFiles = 0
lbyteFiles = 0
duplc = []
matrixc = [[]]
dupl = []
matrixn = [[]]

# It implements a bubble sort comparing only the file names of every file path in alist.
def bubbleSort(alist):
    for passnum in range(len(alist)-1, 0, -1):
        for i in range(passnum):
            name = alist[i]
            nametmp = alist[i+1]
            while '/' in name:
                name = name[name.find('/') + 1:]
            while '/' in nametmp:
                nametmp = nametmp[nametmp.find('/') + 1:]
            if name > nametmp:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp


# This while loop takes every argument and puts them into different queues according to their types. If the argument is
# an option with another string then it combines the two in s string s and appends it to oplist, else if it is an option
# with a single string then it directly appends it to oplist. Else, if the argument is a path that exists, we append it
# to qlist. If it doesn't exist we print a message.
counter = 0
i = 1
while i < n:
    found1 = re.search(r'before|after|match|bigger|smaller|smaller|zip', sys.argv[i])
    found2 = re.search(r'delete|duplcont|duplname|stats|nofilelist', sys.argv[i])
    if found1:
        s = sys.argv[i] + ' ' + sys.argv[i + 1]
        oplist.append(s)
        i = i + 1
    elif found2:
        oplist.append(sys.argv[i])
    else:
        if not os.path.exists(sys.argv[i]):
            print(sys.argv[i] + ' does not exist.')
        else:
            qlist.append(sys.argv[i])
```

```
    i = i + 1

# This while loop checks every option in the oplist and increments the variables with their names to count the number of
# every option. If option is before or after it stores the date information that comes with the option in adate or
# bdate. If it is match then it stores the expression in exp. If the option is bigger or smaller it takes the size
# information and converts it into bytes. If the option is zip it takes the file name and stores it in zipf.
while oplist:
    option = oplist.popleft()

    if option[1:7] == 'before':
        before += 1
        bdates = option[8:]
        if len(bdates) == 15:
            bdate = int(bdates[:8] + bdates[9:])
        else:
            bdate = int(bdates) * 1000000

    if option[1:6] == 'after':
        after += 1
        adates = option[7:]
        if len(adates) == 15:
            adate = int(adates[:8] + adates[9:])
        else:
            adate = int(adates) * 1000000

    if option[1:6] == 'match':
        match += 1
        exp = option[7:]

    if option[1:7] == 'bigger':
        bigger += 1
        size = option[8:]
        if size[-1:] == 'K':
            bnewsize = int(size[:-1]) * 1024
        elif size[-1] == 'M':
            bnewsize = int(size[:-1]) * 1024 * 1024
        elif size[-1] == 'G':
            bnewsize = int(size[:-1]) * 1024 * 1024 * 1024
        else:
            bnewsize = int(size)

    if option[1:8] == 'smaller':
        smaller += 1
        size = option[9:]
        if size[-1:] == 'K':
            snewsize = int(size[:-1]) * 1024
        elif size[-1] == 'M':
            snewsize = int(size[:-1]) * 1024 * 1024
        elif size[-1] == 'G':
            snewsize = int(size[:-1]) * 1024 * 1024 * 1024
        else:
            snewsize = int(size)

    if option[1:7] == 'delete':
        delete += 1

    if option[1:4] == 'zip':
        zip += 1
        zipf = option[5:]

    if option[1:9] == 'duplcont':
        duplcont += 1
```

```
    if option[1:9] == 'duplname':
        duplname += 1

    if option[1:6] == 'stats':
        stats += 1

    if option[1:11] == 'nofilelist':
        nfl += 1

# If any of the options are called more than once it means there is an error and the variable error becomes 1.
if before > 1 or after > 1 or match > 1 or bigger > 1 or smaller > 1 or delete > 1 or zip > 1 or duplcont > 1 or \
        duplname > 1 or stats > 1 or nfl > 1:
    error = 1
# Else if duplname or duplcont is called with delete then it also means an error
elif (duplname == 1 or duplcont == 1) and delete == 1:
    error = 1
# The two elif blocks below controls if there is an error in the format of the dates given with before and after
elif before == 1:
    test = 0
    if len(bdates) == 15:
        if not (bdates[:8].isalnum() and bdates[8] == 'T' and bdates[9:].isalnum()):
            error = 1
    elif len(bdates) == 8:
        if not bdates.isalnum():
            error = 1
    else:
        error = 1
elif after == 1:
    test = 0
    if len(adates) == 15:
        if not (adates[:8].isalnum() and adates[8] == 'T' and adates[9:].isalnum()):
            error = 1
    elif len(adates) == 8:
        if not adates.isalnum():
            error = 1
    else:
        error = 1

# We start tracing the files if there is no error in options.
if error == 0:

    # If there is no paths given in arguments so that the qlist is empty then we add the current directory to qlist
    if not qlist:
        command = 'pwd'
        os.system(command)
        output = subprocess.check_output(command, shell=True)
        qlist.append(output)

    # While there is an element in qlist the loop first finds the items in given directories and adds them to qlist.
    # Then one by one it pops them from the queue to analyze the files.
    while qlist:
        currentdir = qlist.popleft()
        if currentdir[len(currentdir)-1] == '\n':
            currentdir = currentdir[:-1]
        dircontents = os.listdir(currentdir)
        for name in dircontents:
            currentitem = currentdir + "/" + name
            if os.path.isdir(currentitem):
                qlist.append(currentitem)

            else:
                vnumFiles = vnumFiles + 1
                st = os.stat(currentitem)
```

```
            filesize = st.st_size
            vbyteFiles = vbyteFiles + filesize

            name = currentitem
            while '/' in name:
                name = name[name.find('/') + 1:]

            # This big nested if-else structure checks the before, after, match, bigger and smaller options
            # appearing with different combinations. It checks currentitem's modification date, size and name
            # accordingly and appends it to printlist if it passes the tests.

            if before == 1 or after == 1:
                if before == 1 and after == 1:
                    modtime = st.st_mtime
                    times = datetime.datetime.fromtimestamp(modtime).strftime('%Y%m%dT%H%M%S')
                    time = int(times[:8] + times[9:])
                    if adate <= time:
                        if bdate >= time:
                            if bigger == 1 or smaller == 1:

                                if bigger == 1 and smaller == 1:
                                    if snewsize < bnewsize:
                                        error = 1
                                    elif bnewsize <= filesize <= snewsize:
                                        if match == 1:
                                            found = re.search(exp, name)
                                            if found:
                                                printlist.append(currentitem)
                                        else:
                                            printlist.append(currentitem)

                                elif bigger == 1 and smaller == 0:
                                    if filesize >= bnewsize:
                                        if match == 1:
                                            found = re.search(exp, name)
                                            if found:
                                                printlist.append(currentitem)
                                        else:
                                            printlist.append(currentitem)

                                elif bigger == 0 and smaller == 1:
                                    if filesize <= snewsize:
                                        if match == 1:
                                            found = re.search(exp, name)
                                            if found:
                                                printlist.append(currentitem)
                                        else:
                                            printlist.append(currentitem)

                            else:
                                if match == 1:
                                    found = re.search(exp, name)
                                    if found:
                                        printlist.append(currentitem)
                                else:
                                    printlist.append(currentitem)

                elif before == 1 and after == 0:
                    modtime = st.st_mtime
                    times = datetime.datetime.fromtimestamp(modtime).strftime('%Y%m%dT%H%M%S')
                    time = int(times[:8] + times[9:])

                    if bdate >= time:
```

```
            if bigger == 1 or smaller == 1:

               if bigger == 1 and smaller == 1:
                  if snewsize < bnewsize:
                     error = 1
                  elif bnewsize <= filesize <= snewsize:
                     if match == 1:
                        found = re.search(exp, name)
                        if found:
                           printlist.append(currentitem)
                     else:
                        printlist.append(currentitem)

               elif bigger == 1 and smaller == 0:
                  if filesize >= bnewsize:
                     if match == 1:
                        found = re.search(exp, name)
                        if found:
                           printlist.append(currentitem)
                     else:
                        printlist.append(currentitem)

               elif bigger == 0 and smaller == 1:
                  if filesize <= snewsize:
                     if match == 1:
                        found = re.search(exp, name)
                        if found:
                           printlist.append(currentitem)
                     else:
                        printlist.append(currentitem)

            else:
               if match == 1:
                  found = re.search(exp, name)
                  if found:
                     printlist.append(currentitem)
               else:
                  printlist.append(currentitem)

         elif before == 0 and after == 1:
            modtime = st.st_mtime
            times = datetime.datetime.fromtimestamp(modtime).strftime('%Y%m%dT%H%M%S')
            time = int(times[:8] + times[9:])
            if adate <= time:
               if bigger == 1 or smaller == 1:

                  if bigger == 1 and smaller == 1:
                     if snewsize < bnewsize:
                        error = 1
                     elif bnewsize <= filesize <= snewsize:
                        if match == 1:
                           found = re.search(exp, name)
                           if found:
                              printlist.append(currentitem)
                        else:
                           printlist.append(currentitem)

                  elif bigger == 1 and smaller == 0:
                     if filesize >= bnewsize:
                        if match == 1:
                           found = re.search(exp, name)
                           if found:
                              printlist.append(currentitem)
```

```
                    else:
                        printlist.append(currentitem)

                elif bigger == 0 and smaller == 1:
                    if filesize <= snewsize:
                        if match == 1:
                            found = re.search(exp, name)
                            if found:
                                printlist.append(currentitem)
                        else:
                            printlist.append(currentitem)

                else:
                    if match == 1:
                        found = re.search(exp, name)
                        if found:
                            printlist.append(currentitem)
                    else:
                        printlist.append(currentitem)

            else:
                if bigger == 1 or smaller == 1:

                    if bigger == 1 and smaller == 1:
                        if snewsize < bnewsize:
                            error = 1
                        elif bnewsize <= filesize <= snewsize:
                            if match == 1:
                                found = re.search(exp, name)
                                if found:
                                    printlist.append(currentitem)
                            else:
                                printlist.append(currentitem)

                    elif bigger == 1 and smaller == 0:
                        if filesize >= bnewsize:
                            if match == 1:
                                found = re.search(exp, name)
                                if found:
                                    printlist.append(currentitem)
                            else:
                                printlist.append(currentitem)

                    elif bigger == 0 and smaller == 1:
                        if filesize <= snewsize:
                            if match == 1:
                                found = re.search(exp, name)
                                if found:
                                    printlist.append(currentitem)
                            else:
                                printlist.append(currentitem)

                else:
                    if match == 1:
                        found = re.search(exp, name)
                        if found:
                            printlist.append(currentitem)
                    else:
                        printlist.append(currentitem)

# For zip option we imported zipfile and we opened the zipfile with 'w' mode. Since there cannot be more than one
# file with the same name, we check if there exists a file with the same name and we look for the number that the
# filename appears. Then we add the number minus one with the parentheses around it. After updating the name, we
```

```
# write that file to the zip.
if zip == 1:
    if printlist:
        zf = zipfile.ZipFile(zipf, mode='w')
        matrixz = [['', 0]]
        for f in printlist:
            name = f
            while '/' in name:
                name = name[name.find('/') + 1:]
            foundz = 0
            k = 0
            for y in matrixz:
                if y[0] == name:
                    y[1] += 1
                    k = y[1]
                    foundz = 1
            if foundz == 0:
                matrixz.append([name, 1])
                k = 1
            if k == 1:
                zf.write(f, arcname=name)
            else:
                index = name.rfind('.')
                name = name[:index] + '(' + str(k-1) + ')' + name[index:]
                zf.write(f, arcname=name)
        zf.close()

# If duplcont option is called, it means we should list the files with the same content.
# First we open and read every file in printlist, content of every file is stored in data variable. if data is not
# in duplc list, we append data to duplc. Then we append y to its index in matrixc nested list. matrixc is a nested
# list and it is used for keeping the files with same content at the same index. Then we sort every list in matrixc
# according to their filename, after bubblesort, if nofilelist  is not 1 and the length of the list is greater than
# 1, we print every list with same content between the dashes. This list is sorted  alphabetically according to
# their filenames then we print other files under these files. They are not sorted, they are listed according to
# their traversal sequence.
if duplcont == 1:

    for y in printlist:
        with open(y, 'r') as myfile:
            data = myfile.read()
        if data not in duplc:
            duplc.append(data)
            matrixc.append([])
            ind = duplc.index(data)
            matrixc[ind] = []
        matrixc[duplc.index(data)].append(y)
    print('---------------------------------------')
    for x in matrixc:
        bubbleSort(x)
        if nfl != 1:
            if len(x) > 1:
                for b in x:
                    print(b)
                print('---------------------------------------')
    a1 = 0
    for x in matrixc:
        if nfl != 1 and len(x) == 1:
            print(x[0])
            a1 += 1
    if a1 >= 1:
        print('---------------------------------------')

# If duplname option is called, it means we should list the files with the same name. First we find the part that
```

```
# starts with the last '/' in a while loop.If this name is not in list dupl, we append the name to dupl. Then we
# append y to its index in matrixn nested list. matrixn is a nested list and it is used for keeping the files with
# same name at the same index. Then we sort every list in matrixn, after sorting, if nofilelist is not 1 and length
# of every list is greater than, we print every list with same name between dashes. Listed files are listed
# alphabetically according to their path. Then we print other files under these files. They are not sorted, they
# are listed according to their traversal sequence.

if duplname == 1:
    for y in printlist:
        name = y
        while '/' in name:
            name = name[name.find('/') + 1:]
        if name not in dupl:
            dupl.append(name)
            matrixn.append([])
            ind = dupl.index(name)
        matrixn[dupl.index(name)].append(y)
    dupl.sort()
    print('-------------------------------------')
    for m in matrixn[:-1]:
        m.sort()
        for k in dupl:
            name = m[0]
            while '/' in name:
                name = name[name.find('/') + 1:]
            if name == k and len(m) > 1 and nfl != 1:
                for b in m:
                    print(b)
                print('-------------------------------------')
    a1 = 0
    for x in matrixn:
        if nfl != 1 and len(x) == 1:
            a1 += 1
            print(x[0])
    if a1 >= 1:
        print('-------------------------------------')

# If there exists a delete option then we take every file from printlist(if the list is not empty) and add them to a
# string called ccommand with the linux command 'rm' at the beginning. Then by using os.system we execute the
# command.
if delete == 1:
    if printlist:
        dcommand = 'rm '
        for x in printlist:
            dcommand = dcommand + ' ' + x
        os.system(dcommand)

# if nofilelist, duplcont nor duplname is called we simply print the names of the files that are in printlist
if nfl == 0 and duplcont == 0 and duplname == 0:
    for a in printlist:
        print(a)

# If stats option is called, it means we should print the files at the end of files listing output. First we iterate
#  over every string in printlist, we find its size and we add it to lbytesize to find total size of listed files,
# we find the length of printlist and this number is total number of listed files. We declared vnumFiles and
# vbyteFiles earlier and when we traverse the directory we incremented their values. If stats and duplcont are
# called together, we should also write the number of unique files and total size of unique files.We used matrixc
# for finding unique files if stats and dupl are called together, we should also write the number of files with
# unique names.We used matrixn for finding files with unique names.
if stats == 1:
    for x in printlist:
        st = os.stat(x)
        filesize = st.st_size
```

```
            lbyteFiles = lbyteFiles + filesize
            lnumFiles = lnumFiles + 1
        print("total number of files visited:", vnumFiles)
        print("total size of files visited:", vbyteFiles)
        if nfl == 1:
            print('total number of files listed: 0')
            print('total size of files listed: 0')
        else:
            print('total number of files listed:', lnumFiles)
            print('total size of files listed:', lbyteFiles)
        if duplcont == 1:
            if nfl == 1:
                print('total number    of unique files listed: 0')
            else:
                print('total number of unique files listed:', len(matrixc))
            uniquelistsize = 0
            for x in matrixc[:-1]:
                st = os.stat(x[0])
                filesize = st.st_size
                uniquelistsize = uniquelistsize + filesize
            print('total number of unique files in bytes:', uniquelistsize)
        elif duplname == 1:
            print('total number of files with unique names:', len(matrixn)-1)
        print('---------------------------------------')

# If error is not 0 we print an error message and terminate.
else:
    print('Syntax Error')
```