**11/12/15**

"""

Donato, Brandon

bdonato1@binghamton.edu

CS 110 - B57

Jia Yang

Assignment10Ex1

"""

"""

Analysis:

Write a program that repeatedly encrypts or decrypts a text file given the

operation to perform, the text file name, and either the rotation key

(when encrypting) or the rotation key that was used to encrypt (in the case

of decrypting)

Input from keyboard:

fileName (str)

operation (str) - 'E', 'e', 'D', or 'd'

rotationKey(int)

Tasks allocated to functions:

operationValidated() - simple Predicate function

rotationKeyValidated() - simple Predicate function

convertRotationKey()

keepInBounds()

processMessage()

fileNameValidated() - simple Predicate function

makeName() - makes the name for the outfile based upon encryption or

decryption

writeToFile() - writes the encrypted or decrypted text into an outfile

"""

#Imports ----------------------------------------------------------------------

import os.path

#Initialize constants ---------------------------------------------------------

**Page 1 of 7**

# Min and limit ordinals of printable ASCII and total printable ASCII

# characters

PRINTABLE\_ASCII\_MIN = 32

PRINTABLE\_ASCII\_LIMIT = 127

TOTAL\_PRINTABLE\_ASCII\_CHARACTERS = 95

# Allowable rotation key prefixes

KEY\_PREFIX = "-"

# Mapping of valid operations to rotationKey factor

OPERATIONS = {'e':[1,"Encrypted"], 'd':[-1,"Decrypted"]}

# Required file extension

FILE\_EXT = ".txt"

# File processing modes

READ\_MODE = 'r'

WRITE\_MODE = 'w'

# Functions ------------------------------------------------------------------

# Check that requested operation is valid

# param opStr (str) - operation requested

# invoke len()

# invoke str.lower()

# return True when valid, False otherwise (bool)

def operationValidated(opStr):

return len(opStr) == 1 and opStr.lower() in OPERATIONS

# Check that rotation key is of form <digits> or -<digits>

# param rotationKeyStr (str)

# invoke str.isdigit()

# returns: True when valid, False otherwise (bool)

def rotationKeyValidated(rotationKeyStr):

if rotationKeyStr[0] == KEY\_PREFIX:

rotationKeyStr = rotationKeyStr[1:]

return rotationKeyStr.isdigit() and\

int(rotationKeyStr) > 0

**Page 2 of 7**

# Convert rotation key to value usable for requested operation

# param op (str) - operation requested

# param rotationKeyStr (str)

# invoke int()

# invoke str.lower()

# return encryption or decryption rotation key (int)

def convertRotationKey(opStr, rotationKeyStr):

rotationKeyInt = int(rotationKeyStr) \* OPERATIONS[opStr.lower()][0]

return rotationKeyInt

# Perform string modulus operation to prevent processed character

# from going out of bounds

# param ordinal (int)

# returns adjusted ordinal of new character (int)

def keepInBounds(ordinal):

while ordinal >= PRINTABLE\_ASCII\_LIMIT:

ordinal = ordinal - TOTAL\_PRINTABLE\_ASCII\_CHARACTERS

while ordinal < PRINTABLE\_ASCII\_MIN :

ordinal = ordinal + TOTAL\_PRINTABLE\_ASCII\_CHARACTERS

##print(ordinal)

return ordinal

# Encrypt or decrypt message using rotationKey

# param message (str)

# param rotationKey (int)

# invoke keepInBounds()

# return processedMessage (str)

def processMessage(message, rotationKey):

processedMessage = ""

for char in message:

if char != '\n':

ordinal = ord(char)

ordinal += rotationKey

processedMessage += chr(keepInBounds(ordinal))

else:

processedMessage += char

##print(processedMessage)

return processedMessage

**Page 3 of 7**

# Checks that file exists and that extension is .txt

# param name (str) - file name

# invoke isFile() from module os.path and endswith()

# return True when valid, False otherwise (bool)

def fileNameValidated(name):

return os.path.isfile(name) and name.endswith(FILE\_EXT)

# Generates output file name from input file name,

# operation requested and rotation key

# param fileName (str) - input file name

# param operation (str)

# param rotationKey (int) - converted key

# invoke str.split(), str.replace() and str.join()

# return output file name (str)

def makeName(fileName, operation, rotationKey):

nameList = fileName.split(".")

nameList[0] = nameList[0].replace(OPERATIONS['e'][1], "")

nameList[0] = nameList[0].replace(OPERATIONS['d'][1], "")

nameList[0] += (OPERATIONS[operation][1] + str(rotationKey))

return ".".join(nameList)

# Writes a string of text to an outfile given

# param text (str) - string to be written

# param outFile - the file where the string is being written

# invoke write()

# returns nothing

def writeToFile(text, outFile):

for line in text:

outFile.write(line)

# Main -----------------------------------------------------------------------

# Gets plain text or cipher code, operation requested (encrypt or decrypt),

# and rotation key for Caesar cipher

# Generates cipher code or plain text

def main():

# Describe program

print("This program encrypts or decrypts entire files " + \

"using a Caesar cipher")

**Page 4 of 7**

# Priming read, file validation, and repeat

ilename = input(“What is the file you would like to be processed\nOr\

press <Enter> to quit: “)

while not fileNameValidated(ilename):

print(“\nThe file you’ve given isn’t a .txt file or doesn’t exist.\

\nPlease enter another file name and try again.\n”)

ilename = input(“What is the file you would like to be processed : “)

while ilename:

try: # Outer try for opening the given file

inFile = open(ilename, READ\_MODE)

try: # Inner try for processing the materials in the file

message = inFile.read()

##print(message)

try: # Outer try for encryption of the file

# Looks to see if the file wants to be encrypted or decrypted and

# validates it

print(“Would you like to encrypt or decrypt the following file?”)

operation = input(“Enter <E> for encryption or <D> for decryption : “)

while not operationValidated(operation):

print(“That is not a valid input. Please try again.”)

operation =\

input(“Enter <E> for encryption or <D> for decryption : “)

# Processes and validates the encryption key

rotationKey = input(“Enter a rotation key or the rotation key\

that was used: “)

while not rotationKeyValidated(rotationKey):

print(“That is not a valid input. Please try again.”)

rotationKey = input(“Enter a rotation key or the rotation key\

that was used: “)

rotationKey = convertRotationKey(operation, rotationKey)

# Encrypts or decrypts the contents of a file

newMessage = processMessage(message, rotationKey)

##print(newMessage)

**Page 5 of 7**

# Makes the outfile name and opens the outfile, preparing it

# to be written in

outFileName = makeName(ilename,operation,rotationKey)

outFile = open(outFileName, WRITE\_MODE)

##print(outFile)

try: # Inner try for writing the encrypted or decrypted message

writeToFile(newMessage,outFile)

# Inner exception handler for outfile processing

# Will close the file whether or not an exception has been raised

except OSError as err:

print(“\nAn exception was raised while opening the given file. Try\

a different file.\n”)

except TypeError as err:

print(“\nThere was a problem with writing the data. Is the input\

corrupted or in the wrong format?\n”)

except Exception as err:

print(“There was an error. “ + str(err) + ‘\n’)

finally:

outFile.close()

# Outer exception handler for outfile opening and encryption

except OSError as err:

print(“An exception was raised while opening the given file. Try\

a different file.\n”)

except Exception as err:

print(“There was an error. “ + str(err) + ‘\n’)

# Inner exception handler for infile processing

# Will close the file whether or not an exception has been raised

except OSError as err:

print(“An exception was raised while opening the given file. Try a\

different file.\n”)

except ValueError as err:

print(“There was a problem processing the data. It could be in the\

wrong folder or the data could be corrupted. \n”)

except Exception as err:

print(“There was an error. “ + str(err) + ‘\n’)

finally:

inFile.close()

**Page 6 of 7**

# Outer exception handler for infile opening

except FileNotFoundError as err:

print("\nThis file was not found.\nDoes this file exist?\n"\

+ str(err) + '\n')

except OSError as err:

print("An exception was raised while opening the given file. Try a\

different file.\n")

except Exception as err:

print("There was an error. " + str(err) + '\n')

# Notify the user that the message has been processed

print("The file has been processed.\n")

# Continuation read

fileName = input("Enter a file to be processed or press <Enter>\

to leave: ")

main()

**Page 7 of 7**