**10/22/15**

"""

Donato, Brandon

bdonato1@binghamton.edu

CS 110 - B57

Jia Yang

Assignment7Ex1

"""

"""

Analysis:

Write a program that repeatedly encrypts or decrypts a message given the

operation to perform and either the rotation key (when encrypting) or the

rotation key that was used to encrypt (in the case of decrypting)

Output to monitor:

newMessage (str)

Input from keyboard:

message (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()

"""

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

OPERATIONS = "ed"

ENCRYPT = 1

DECRYPT = -1

# 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

**Page 1 of 4**

# Allowable rotation key prefixes

KEY\_PREFIX = "-"

# 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

# Convert rotation key to value usable for requested operation

# param op (str) - operation requested

# param rotationKeyStr (str)

# invoke int()

# return encryption or decryption rotation key (int)

def convertRotationKey(opStr, rotationKeyStr):

if opStr == 'e' or opStr == 'E':

rotationKeyInt = ENCRYPT \* int(rotationKeyStr)

else:

rotationKeyInt = DECRYPT \* int(rotationKeyStr)

#print(rotationKeyInt)

return rotationKeyInt

**Page 2 of 4**

# 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):

if ordinal >= PRINTABLE\_ASCII\_LIMIT:

while ordinal >= PRINTABLE\_ASCII\_LIMIT:

ordinal = ordinal - TOTAL\_PRINTABLE\_ASCII\_CHARACTERS

elif ordinal < PRINTABLE\_ASCII\_MIN :

while ordinal < PRINTABLE\_ASCII\_MIN :

ordinal = ordinal + TOTAL\_PRINTABLE\_ASCII\_CHARACTERS

else:

ordinal = ordinal

#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:

ordinal = ord(char)

ordinal += rotationKey

processedMessage += chr(keepInBounds(ordinal))

#print(processedMessage)

return processedMessage

# 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 messages " + \

"using a Caesar cipher")

**Page 3 of 4**

# Priming read and repeat

message = input("What is the message you would like to be processed : ")

while message:

# Get remaining inputs, validate and convert as necessary

print("Would you like to encrypt or decrypt the following\

message?")

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 : ")

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)

#print(type(rotationKey))

# Encrypt or decrypt contents of file

newMessage = processMessage(message, rotationKey)

# Display result

print("The message is: %s" % (newMessage,))

print("\n")

# Continuation read

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

to leave: ")

main()

**Page 4 of 4**