Walter Guyer

Wnguyer

800093011

Programming Assignment #1

6 February 2019

**STATUS SHEET**

* All aspects of program were tested and should be functioning properly

**DOCUMENTATION**

* Written and compiled using Python3
* Tested in Windows and Mac OSX environment using Python3 compiler
* Libraries:
  + **Re.py**
    - Python Standard Library – Regular expression operations
  + **Os.py**
    - Python Standard Library – Miscellaneous operating system interfaces
  + **Sys.py**
    - Python Standard Library – System-specific parameters and functions

# Walter Guyer

# wnguyer

# 800093011

# Programming Assignment #1

# 6 FEB 2019

# wguyerDES.py should be run using command "python3 wguyerDES.py" from

# directory with plaintext.txt file containing text to be encrypted.

# Program will create an output.txt file containing output from each

# stage of encryption and decryption along with a final line at the bottom

# with text after it has been encrypted, then decrypted.

import re

import os

import sys

#Permuted Choice 1

PC\_1 = [57,49,41,33,25,17,9,1,58,50,42,34,26,18,10,2,59,51,43,35,27,19,11,3,60,52,44,36,63,55,47,39,31,23,15,7,62,54,46,38,30,22,14,6,61,53,45,37,29,21,13,5,28,20,12,4]

#Shift Sequence

SHIFTS = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

#Permuted Choice 2

PC\_2 = [14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,37,47,55,30,40,51,45,33,48,44,49,39,56,34,53,46,42,50,36,29,32]

#Initial Permutation

IP = [58,50,42,34,26,18,10,2,60,52,44,36,28,20,12,4,62,54,46,38,30,22,14,6,64,56,48,40,32,24,16,8,57,49,41,33,25,17,9,1,59,51,43,35,27,19,11,3,61,53,45,37,29,21,13,5,63,55,47,39,31,23,15,7]

#Inverse Initial Permutation

IP\_INVERSE = [40,8,48,16,56,24,64,32,39,7,47,15,55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,49,17,57,25]

#E Bit-selection

E = [32,1,2,3,4,5,4,5,6,7,8,9,8,9,10,11,12,13,12,13,14,15,16,17,16,17,18,19,20,21,20,21,22,23,24,25,24,25,26,27,28,29,28,29,30,31,32,1]

#Permutation function P

P = [16,7,20,21,29,12,28,17,1,15,23,26,5,18,31,10,2,8,24,14,32,27,3,9,19,13,30,6,22,11,4,25]

#S-Boxes

S = [

#S1

[

[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7],

[0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8],

[4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0],

[15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13]

],

#S2

[

[15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10],

[3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5],

[0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15],

[13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9]

],

#S3

[

[10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8],

[13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1],

[13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7],

[1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12]

],

#S4

[

[7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15],

[13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9],

[10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4],

[3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14]

],

#S5

[

[2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9],

[14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6],

[4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14],

[11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3]

],

#S6

[

[12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11],

[10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8],

[9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6],

[4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13]

],

#S7

[

[4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1],

[13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6],

[1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2],

[6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12]

],

#S8

[

[13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7],

[1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2],

[7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8],

[2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11]

]

]

def permutedChoice1(key64):

key56 = ""

for i in PC\_1:

key56 += key64[i-1]

return key56

def permutedChoice2(key56):

key48 = ""

for i in PC\_2:

key48 += key56[i-1]

return key48

def shiftBits(i,key):

shiftedKey = key[SHIFTS[i]:] + key[:SHIFTS[i]]

return shiftedKey

def createKeys(key64):

#list of subkeys

keys = []

#Perform Permuted Choice 1

key56 = permutedChoice1(key64)

print("The key after permute is: " + key56 + "\n")

#Split bits

c = key56[:28]

d = key56[28:]

#16 rounds of keys

for i in range(16):

#Shift bits

Cn = shiftBits(i,c)

Dn = shiftBits(i,d)

#Permuted Choice 2

keys.append(permutedChoice2(Cn + Dn))

print("Key " + str(i+1) + " is " + keys[i])

c = Cn

d = Dn

return keys

def textToBin(text):

binString = bin(int.from\_bytes(text.encode(), 'big'))

return binString[2:]

def stripText(text):

text = re.sub("[^A-Za-z0-9]+", "", text)

return text

def preProcess(text):

check = 0

#Check if last block will need filled

if len(text) % 64 != 0:

check = 1

#Initialize blocks to be filled using check to account for unfilled box

textBlocks = [""] \* (int(len(text)/64) + check)

#Fill blocks with text

for i in range(len(text)):

x = int(i/64)

textBlocks[x] += text[i]

#Fill 0's in end block if necessary

for i in range(len(textBlocks)):

while len(textBlocks[i]) < 64:

textBlocks[i] += "0"

return textBlocks

def initialPermutation(block64):

permutedBlock = ""

for i in IP:

permutedBlock += block64[i-1]

return permutedBlock

def expansion(block32):

expandedBlock = ""

for i in E:

expandedBlock += block32[i-1]

return expandedBlock

def xOR(block1,block2):

xorBlock = ""

for i in range(len(block1)):

if block1[i] == block2[i]:

xorBlock += "0"

else:

xorBlock += "1"

return xorBlock

def sBox(block48):

block32 = ""

sixBlocks = [""] \* 8

for i in range(8):

for j in range(6):

sixBlocks[i] += block48[j+(6\*i)]

for i in range(8):

block32 += bin(S[i][int((sixBlocks[i][0]+sixBlocks[i][5]),2)][int((sixBlocks[i][1:5]),2)])[2:].zfill(4)

return block32

def permutationP(block48):

permuted48 = ""

for i in P:

permuted48 += block48[i-1]

return permuted48

def inverseP(block64):

permutedBlock = ""

for i in IP\_INVERSE:

permutedBlock += block64[i-1]

return permutedBlock

def encryptFunction(encryptText,subkeyList):

#Text Processing

encryptText = stripText(encryptText)

encryptText = "0" + textToBin(encryptText)

preBlocks = preProcess(encryptText)

print("Data after preprocessing: \n")

for i in preBlocks:

for x in range(0,len(i),8):

print(i[x:x+8])

print()

for i in range(len(preBlocks)):

preBlocks[i] = initialPermutation(preBlocks[i])

print("Initial permuation result: \n" + preBlocks[i] + "\n")

Li\_1 = preBlocks[i][:32]

Ri\_1 = preBlocks[i][32:]

#Encryption Cycle

for x in range(16):

print("Iteration: " + str(x+1))

print("L\_i-1:\n" + Li\_1 + "\n")

print("R\_i-1:\n" + Ri\_1 + "\n")

temp = Ri\_1

Ri\_1E = expansion(Ri\_1)

print("Expansion permutation:\n" + Ri\_1E + "\n")

Ri\_1E = xOR(Ri\_1E,subkeyList[x])

print("XOR with key:")

for y in range(0,len(Ri\_1E),6):

print(Ri\_1E[y:y+6])

print()

Ri\_1E = sBox(Ri\_1E)

print("S-box substitution:\n" + Ri\_1E + "\n")

Ri\_1E = permutationP(Ri\_1E)

print("P-box substitution:\n" + Ri\_1E + "\n")

Ri\_1 = xOR(Li\_1,Ri\_1E)

print("XOR with L\_i-1 (This is R\_i):\n" + Ri\_1E + "\n")

Li\_1 = temp

print("End iteration: " + str(x+1) + "\n\n")

preBlocks[i] = inverseP(Ri\_1 + Li\_1)

print("Final permutation:\n" + preBlocks[i] + "\n")

return preBlocks

#Same as encryption w/o preprocessing of text, w/ reverse subkey list

def decryptFunction(encryptedBlocks,subkeyList):

for i in encryptedBlocks:

for x in range(0,len(i),8):

print(i[x:x+8])

print()

for i in range(len(encryptedBlocks)):

encryptedBlocks[i] = initialPermutation(encryptedBlocks[i])

print("Initial permuation result: \n" + encryptedBlocks[i] + "\n")

Li\_1 = encryptedBlocks[i][:32]

Ri\_1 = encryptedBlocks[i][32:]

for x in range(16):

print("Iteration: " + str(x+1))

print("L\_i-1:\n" + Li\_1 + "\n")

print("R\_i-1:\n" + Ri\_1 + "\n")

temp = Ri\_1

Ri\_1E = expansion(Ri\_1)

print("Expansion permutation:\n" + Ri\_1E + "\n")

Ri\_1E = xOR(Ri\_1E,subkeyList[x])

print("XOR with key:")

for y in range(0,len(Ri\_1E),6):

print(Ri\_1E[y:y+6])

print()

Ri\_1E = sBox(Ri\_1E)

print("S-box substitution:\n" + Ri\_1E + "\n")

Ri\_1E = permutationP(Ri\_1E)

print("P-box substitution:\n" + Ri\_1E + "\n")

Ri\_1 = xOR(Li\_1,Ri\_1E)

print("XOR with L\_i-1 (This is R\_i):\n" + Ri\_1E + "\n")

Li\_1 = temp

print("End iteration: " + str(x+1) + "\n\n")

encryptedBlocks[i] = inverseP(Ri\_1 + Li\_1)

print("Final permutation:\n" + encryptedBlocks[i] + "\n")

return encryptedBlocks

#Open plaintext.txt file

with open(os.path.join(os.getcwd(), "plaintext.txt"), "r") as f:

textToEncrypt = f.read()

f.close()

#Print text from file and grab password from user

print("Text to encrypt: " + textToEncrypt)

key64 = input("Password: ")

#Create output.txt file and change stdout to file

sys.stdout = open(os.path.join(os.getcwd(), "output.txt"), "w")

print("Text to encrypt: " + textToEncrypt + "\nPassword: " + key64)

#Subkey generation and encryption call

subKeys = createKeys(textToBin(key64))

encryptedText = encryptFunction(textToEncrypt,subKeys)

#Prints encrypted blocks

for i in encryptedText:

for x in range(0,len(i),8):

print(i[x:x+8])

print("\n")

#Decryption Call w/ reversed subkey list

decryptedTextBlocks = decryptFunction(encryptedText,list(reversed(subKeys)))

#Creates string representation of decrypted blocks

decryptedText = ""

for i in decryptedTextBlocks:

decryptedText += i

decryptedText = "0b" + decryptedText

print(stripText(int(decryptedText, 2).to\_bytes((int(decryptedText, 2).bit\_length() + 7) // 8, 'big').decode()))