[암호론 과제]

과목명	암호론	담당 교수님	박태룡
전 공	컴퓨터공학과	학 번	2020305082
제출일	2023.06.14	이 름	하정원

1. 소스 코드

```
import random

po_1= [49,42,35,28,21,14,7,0,
       50,43,36,29,22,15,8,1,
       51,44,37,30,23,16,9,2,
       52,45,38,31,55,48,41,34,
       27,20,13,6,54,47,40,33,
       26,19,12,5,53,46,39,32,
       25,18,11,4,24,17,10,3]

shift_n = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

po_2=[13,16,10,23,0,4,2,27,
      14,5,20,9,22,18,11,3,
      25,7,15,6,26,19,12,1,
      40,51,30,36,46,54,29,39,
      50,44,32,47,43,48,38,55,
      33,52,45,41,49,35,28,31]

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

```
class Round_Func:
    def __init__(self, R, key, po1, po2, shift_n, s_box, p_box, count):
        self.__po1 = po_1
        self.__po2 = po_2
        self.__shift_n = shift_n
        self.__sbox = s_box
        self.__pbox = p_box
        self.__r = R
        self.__key = key
        self.current_n = count
        self.result = ""

    def expansion_R(self) :
        list_byte = []
        e_byte = []
        n = 0
        m = 0
        for i in self.__r:
            list_byte.append(i)
            n+=1
            m+=1
            if(n == 4):
                byte_t = ''.join(list_byte)
                if(m == 32):
                    head_text = self.__r[m-5]
                    tail_text = self.__r[0]
                else:
                    head_text = self.__r[m-5]
                    tail_text = self.__r[m+1]
                e_byte.append(head_text+byte_t+tail_text)
                n = 0
                list_byte = []

        return ''.join(e_byte)
```

```
def StrToByte(text):
    byte_array = bytearray(text.encode('ascii'))
    str_list = []
    for i in byte_array:
        str_list.append("0"+str(format(i,"b")))
    return ''.join(str_list)

def key_generator(n):
    key = format(random.getrandbits(n),'b')
    return key
key = key_generator(64).zfill(64)

def ByteToStr(text_byte):
    list_byte = []
    text_list = []
    n=0
    for i in text_byte:
        list_byte.append(i)
        n+=1
        if(n==8):
            bit_t = ''.join(list_byte)
            str_b = "0b" + bit_t
            text_list.append(chr(int(str_b,2)))
            n=0
            list_byte = []
    return ''.join(text_list)
```

```
def shift_left(self, n, d, length):
    result = ( (n<<d) & (2**length -1) ) | (n >>(length - d))
    return result

def subkey_generator(self):
    subkey_po1 = [0] * len(self.__po1)
    subkey_po2 = [0] * len(self.__po2)
    n = 0
    for i in self.__po1:
        subkey_po1[n] = self.__key[i]
        n += 1
    subkey_po = ''.join(subkey_po1)

    o_subkey = subkey_po[:int(len(subkey_po1)/2)]
    d_subkey = subkey_po[int(len(subkey_po1)/2):]
    len_subkey = len(o_subkey)

    o_shift = bin(self.shift_left(int(o_subkey,2), int(self.__shift_n[self.current_n]), len_subkey))[2:].zfill(len_subkey)
    d_shift = bin(self.shift_left(int(d_subkey,2), int(self.__shift_n[self.current_n]), len_subkey))[2:].zfill(len_subkey)
    subkey_shift = o_shift + d_shift

    n = 0
    for i in self.__po2:
        subkey_po2[n] = subkey_shift[i]
        n+=1

    return ''.join(subkey_po2)

def s_box(self, subkey_po2):
    n = 0
    bit_list = []
    s_box = []
    for i in subkey_po2 :
        bit_list.append(i)
        n += 1
        if (n % 6) ==0 :
            s_val = bin(self.__sbox[(n//6)-1][int(bit_list[0]+bit_list[5],2)]
                        [int(bit_list[1]+bit_list[2]+bit_list[3],2)])[2:].zfill(6)
            s_box.append(s_val)
            bit_list = []

    return ''.join(s_box)
```

```
def s_box(self, subkey_po2):
    n = 0
    bit_list = []
    s_box = []
    for i in subkey_po2 :
        bit_list.append(i)
        n += 1
        if (n % 6) ==0 :
            s_val = bin(self.__sbox[(n//6)-1][int(bit_list[0]+bit_list[5],2)]
                        [int(bit_list[1]+bit_list[2]+bit_list[3],2)])[2:].zfill(6)
            s_box.append(s_val)
            bit_list = []

    return ''.join(s_box)

def p_box(self, s_box):
    n = 0
    subkey = [0]*len(self.__pbox)
    for i in  self.__pbox:
        subkey[n] = s_box[i]
        n += 1
    return ''.join(subkey)
```

```
    def active_func(self):
        e_r = self.expansion_R()
        s_p = self.subkey_generator()
        subkey_xor = bin(int(e_r,2) ^ int(s_p,2))[2:].zfill(48)
        subkey_sbox = self.s_box(subkey_xor)
        self.result = self.p_box(subkey_sbox)

class DES:
    def __init__ (self, plainText, key, num_round, po_1, po_2, shift_n, s_box, p_box):
        self.__plaintext = plainText
        self.__key = key
        self.num_round = num_round
        self.resulttext = ''
        self.__po1 = po_1
        self.__po2 = po_2
        self.__shift_n = shift_n
        self.__sbox = s_box
        self.__pbox = p_box

    def encryption(self):
        l_text = self.__plaintext[:32]
        r_text = self.__plaintext[32:]

        for i in range(self.num_round):
            r_tmp = r_text

            f_obj = Round_Func(r_text, self.__key, self.__po1, self.__po2, self.__shift_n,
                       self.__sbox, self.__pbox, i)
            f_obj.active_func()
            result_round = bin(int(l_text,2) ^ int(f_obj.result,2))[2:].zfill(32)

            r_text = result_round
            l_text = r_tmp

        self.resulttext = r_text + l_text
```

```
plaintext = "123456ab"

name = StrToByte(plaintext)

print("-----------------------------------------------------------")
print("<<DES 암호화>>")


print("ㅔnㅔn평문 :", plaintext)

obj = DES(name, key, 16, po_1, po_2, shift_n, s_box, p_box)
obj.encryption()
print("암호문 :",ByteToStr(obj.resulttext))

print("-----------------------------------------------------------")
```

2. 결과

```
------------------------------------------------------------------
<<DES 암호화>>


평문 : 123456ab
암호문 : □ B ạbYÙÀN
------------------------------------------------------------------
```