

EXP NO: 1a  
DATE: 27/01/24

## CAESER CIPHER

### AIM:

To write a c program to implement caesar cipher.

### ALGORITHM:

1. Create a list or string representing the alphabet (typically, the 26 letters of the English alphabet in order).
2. Choose the shift value (an integer) that determines how many positions each letter in the plaintext will be shifted.
3. Input the plaintext message that you want to encrypt.
4. Convert the plaintext message to lowercase to ensure uniformity (if case sensitivity is not required).
5. For each character in the plaintext: If the character is a letter:
  - \*Find the position (index) of the character in the alphabet.
  - \*Compute the new position by adding the shift value to the current position, then take the result modulo 26 (the length of the alphabet) to handle wrap-around.
  - \*Replace the character with the letter at the new position.
  - \*If the character is not a letter, leave it unchanged.
6. Append each transformed character to a new string or list to form the ciphertext.
7. Combine the list of characters into a single string (if necessary) and output the cipher.

**PROGRAM:**

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <ctype.h>

int main()
{
    char message[500], c;
    int i;
    int key;

    printf("Enter a message to encrypt: ");
    scanf("%[^\n]", message); // Read the whole line including spaces

    printf("Enter key: ");
    scanf("%d", &key);

    for (i = 0; message[i] != '\0'; i++) {
        c = message[i];

        // Encrypt alphabets (both lowercase and uppercase)
        if (isalpha(c)) {
            if (islower(c)) {
                c = (c - 'a' + key) % 26 + 'a';
            } else {
                c = (c - 'A' + key) % 26 + 'A';
            }
        } else { // Encrypt special characters
            c = (c + key) % 128;
        }

        message[i] = c;
    }

    printf("Encrypted message: %s\n", message);

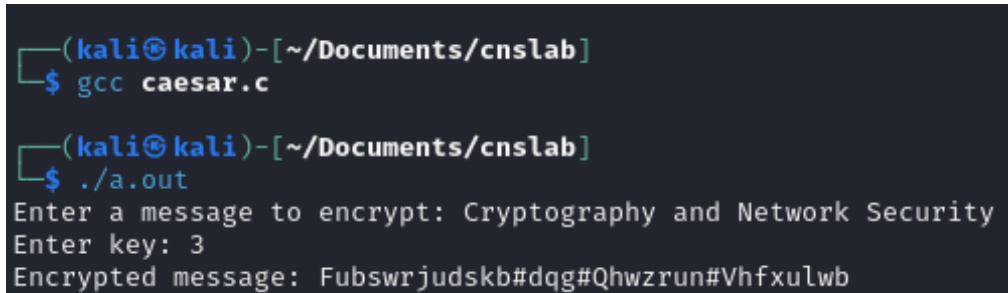
    printf("*****Decryption*****");
    char message[500], c;
    int i;
    int key;
```

```
printf("Enter a message to decrypt: ");
scanf("%[^\n]", message); // Read the whole line including spaces
printf("Enter key: ");
scanf("%d", &key);
for (i = 0; message[i] != '\0'; i++) {
    c = message[i];
    // Decrypt alphabets (both lowercase and uppercase)
    if (isalpha(c)) {
        if (islower(c)) {
            c = (c - 'a' - key + 26) % 26 + 'a';
        } else {
            c = (c - 'A' - key + 26) % 26 + 'A';
        }
    } else { // Decrypt special characters
        c = (c - key + 128) % 128;
    }

    message[i] = c;
}
printf("Decrypted message: %s\n", message);

return 0;
}
```

OUTPUT:



```
(kali㉿kali)-[~/Documents/cnslab]
$ gcc caesar.c

(kali㉿kali)-[~/Documents/cnslab]
$ ./a.out
Enter a message to encrypt: Cryptography and Network Security
Enter key: 3
Encrypted message: Fubswrjudskb#dqg#Qhwzrun#Vhfxulwb
```

RESULT:

Thus a C program was implemented to demonstrate Caesar Cipher.

EXP NO: 1b

DATE: 03/02/2024

### PLAYFAIR CIPHER

AIM:

To write a python program to implement playfair cipher.

ALGORITHM:

1. Construct a 5x5 grid to hold the key square for the cipher.
2. Select a keyword or key phrase to populate the grid.
3. Write the keyword into the grid, without repeating any letters.  
Combine the letters 'I' and 'J' to fit the alphabet into 25 cells.
4. After the keyword, fill in the remaining cells of the grid with the rest of the alphabet in order, excluding the letter 'J'.
5. Divide the plaintext message into digraphs (pairs of two letters). If a pair contains the same letter (e.g., "LL"), insert an 'X' between them to separate the letters. If the message length is odd, add an 'X' at the end to make it even.
6. For each digraph: If both letters are in the same row, replace them with the letters immediately to their right (wrap around to the start of the row if necessary).
7. If both letters are in the same column, replace them with the letters immediately below them (wrap around to the top of the column if necessary). If the letters form a rectangle, replace them with the letters on the same row but at the other pair of corners of the rectangle.
8. Combine the decrypted digraphs to reconstruct the plaintext message. Remove any extra 'X' characters that were added during encryption.

**PROGRAM:**

```
def toLowerCase(text):  
    return text.lower()
```

# Function to remove all spaces in a string

```
def removeSpaces(text):  
    newText = ""  
    for i in text:  
        if i == " ":  
            continue  
        else:  
            newText = newText + i  
    return newText
```

# Function to group 2 elements of a string  
# as a list element

```
def Diagraph(text):  
    Diagraph = []  
    group = 0  
    for i in range(2, len(text), 2):  
        Diagraph.append(text[group:i])  
  
        group = i  
    Diagraph.append(text[group:])  
    return Diagraph
```

# Function to fill a letter in a string element  
# If 2 letters in the same string matches

```
def FillerLetter(text):  
    k = len(text)  
    if k % 2 == 0:
```

```

    for i in range(0, k, 2):
        if text[i] == text[i+1]:
            new_word = text[0:i+1] + str('x') + text[i+1:]
            new_word = FillerLetter(new_word)
            break
        else:
            new_word = text
    else:
        for i in range(0, k-1, 2):
            if text[i] == text[i+1]:
                new_word = text[0:i+1] + str('x') + text[i+1:]
                new_word = FillerLetter(new_word)
                break
            else:
                new_word = text
    return new_word

```

```

list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',
        'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

```

# Function to generate the 5x5 key square matrix

```

def generateKeyTable(word, list1):
    key_letters = []
    for i in word:
        if i not in key_letters:
            key_letters.append(i)

    compElements = []
    for i in key_letters:
        if i not in compElements:
            compElements.append(i)
    for i in list1:
        if i not in compElements:
            compElements.append(i)

```

```
matrix = []
while compElements != []:
    matrix.append(compElements[:5])
    compElements = compElements[5:]

return matrix


def search(mat, element):
    for i in range(5):
        for j in range(5):
            if(mat[i][j] == element):
                return i, j


def encrypt_RowRule(matr, e1r, e1c, e2r, e2c):
    char1 = ""
    if e1c == 4:
        char1 = matr[e1r][0]
    else:
        char1 = matr[e1r][e1c+1]

    char2 = ""
    if e2c == 4:
        char2 = matr[e2r][0]
    else:
        char2 = matr[e2r][e2c+1]

    return char1, char2


def encrypt_ColumnRule(matr, e1r, e1c, e2r, e2c):
    char1 = ""
    if e1r == 4:
        char1 = matr[0][e1c]
    else:
```

```

        char1 = matr[e1r+1][e1c]

    char2 = "
    if e2r == 4:
        char2 = matr[0][e2c]
    else:
        char2 = matr[e2r+1][e2c]

    return char1, char2
def encrypt_RectangleRule(matr, e1r, e1c, e2r, e2c):
    char1 = "
    char1 = matr[e1r][e2c]

    char2 = "
    char2 = matr[e2r][e1c]

    return char1, char2
def encryptByPlayfairCipher(Matrix, plainList):
    CipherText = []
    for i in range(0, len(plainList)):
        c1 = 0
        c2 = 0
        ele1_x, ele1_y = search(Matrix, plainList[i][0])
        ele2_x, ele2_y = search(Matrix, plainList[i][1])

        if ele1_x == ele2_x:
            c1, c2 = encrypt_RowRule(Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
            # Get 2 letter cipherText
        elif ele1_y == ele2_y:
            c1, c2 = encrypt_ColumnRule(Matrix, ele1_x, ele1_y, ele2_x,
ele2_y)
        else:
            c1, c2 = encrypt_RectangleRule(
                Matrix, ele1_x, ele1_y, ele2_x, ele2_y)

        cipher = c1 + c2
        CipherText.append(cipher)

```



```

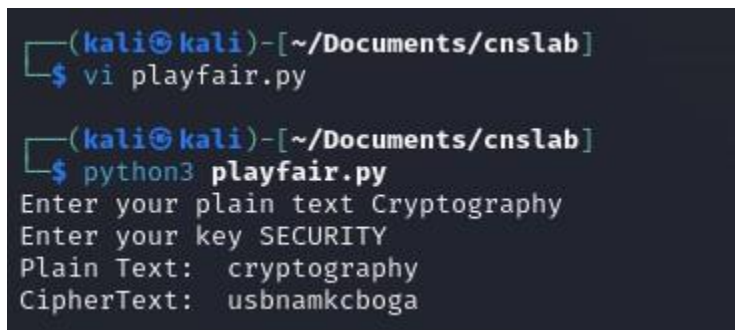
    return CipherText
text_Plain = input("Enter your plain text ")
text_Plain = removeSpaces(toLowerCase(text_Plain))
PlainTextList = Diagraph(FillerLetter(text_Plain))
if len(PlainTextList[-1]) != 2:
    PlainTextList[-1] = PlainTextList[-1]+'z'
key = input("Enter your key ")
key = toLowerCase(key)
Matrix = generateKeyTable(key, list1)

print("Plain Text: ", text_Plain)
CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)

CipherText = ""
for i in CipherList:
    CipherText += i
print("CipherText: ", CipherText)

```

OUTPUT:



```

(kali@kali)-[~/Documents/cnslab]
$ vi playfair.py

(kali@kali)-[~/Documents/cnslab]
$ python3 playfair.py
Enter your plain text Cryptography
Enter your key SECURITY
Plain Text:  cryptography
CipherText:  usbnamkcboga

```

RESULT:

Thus a python program has been implemented to demonstrate Playfair Cipher.

EXP NO: 1c

DATE: 10/02/2024

### RAIL FENCE CIPHER

AIM:

To write a python program to implement Rail Fence Cipher.

ALGORITHM:

1. Decide the number of "rails" (rows) to use for the encryption.
2. Write the plaintext in a zigzag pattern across the chosen number of rails.
3. Start at the top rail: Begin writing the first letter of the plaintext at the top rail.
4. Move diagonally downwards: Continue writing each subsequent letter moving diagonally down to the next rail.
5. Reach the bottom rail: When the bottom rail is reached, reverse direction and start moving diagonally upwards.
6. Move diagonally upwards: Continue writing each subsequent letter moving diagonally up to the previous rail.
7. Repeat the process: Repeat the zigzag writing process until all letters of the plaintext are written in the pattern.
8. Read the ciphertext by sequentially reading the letters row by row from the top rail to the bottom rail.
9. To decrypt, determine the zigzag pattern used by the number of rails. Fill the rails with placeholders to match the zigzag pattern.

## PROGRAM:

```
def main():
    text = input('Input Text : ')
    rows = int(input('Input Rows : '))
    text = text.replace(' ', '')

    while True:
        chc = input('1.Encrypt\n2.Decrypt\nEnter your choice: ')
        if chc in ['0', '1']:
            break
        print('Choose 0 / 1')

    #print(len(text))
    if int(chc):
        arr = [[ ' ' for y in range(len(text))] for x in range(rows)]
        #[ print(row) for row in arr ]

        dir_down = None
        row, col = 0 , 0
        for i in range(len(text)):
            if row == 0: dir_down = True
            if row == rows - 1: dir_down = False

            arr[row][col] = '*'
            col += 1

            if dir_down: row += 1
            else: row -= 1

        #print('\n\n')
        #[ print(row) for row in arr ]
        count = 0
        for row in arr:
            for i in range(len(row)):
                if row[i] == '*':
                    row[i] = text[count]
```

```
        count += 1

#print('\n\n')
#[ print(row) for row in arr ]

result = []
row, col = 0, 0
for i in range(len(text)):

    if row == 0: dir_down = True
    if row == rows-1: dir_down = False

    if (arr[row][col] != '*'):
        result.append(arr[row][col])
        col += 1

    if dir_down: row += 1
    else: row -= 1

print(" ".join(result).strip())
else:
    arr = [ [] for x in range(rows)]
    #print(arr)
    count = 0
    finish = False

    while True:
        for j in range(0,rows-1):
            arr[j].append(text[count])
            count += 1

            if count >= len(text):
                finish = True
                break

        if finish :
            break
```

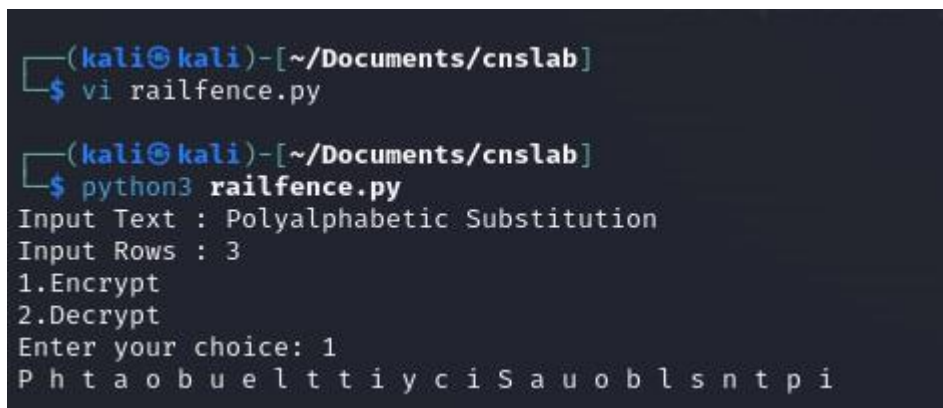
```
for k in range(rows - 1 ,0,-1):
    arr[k].append(text[count])
    count += 1

if count >= len(text):
    finish = True
    break

if finish :
    break
print(arr)
```

main()

OUTPUT:

A terminal window with a dark background and light blue text. The prompt is '(kali@kali)-[~/Documents/cnslab]'. The user enters '\$ vi railfence.py'. The prompt is '(kali@kali)-[~/Documents/cnslab]'. The user enters '\$ python3 railfence.py'. The program outputs 'Input Text : Polyalphabetic Substitution', 'Input Rows : 3', a menu '1.Encrypt', '2.Decrypt', and 'Enter your choice: 1'. The final output is 'P h t a o b u e l t t i y c i S a u o b l s n t p i' with spaces between the characters.

```
(kali@kali)-[~/Documents/cnslab]
$ vi railfence.py

(kali@kali)-[~/Documents/cnslab]
$ python3 railfence.py
Input Text : Polyalphabetic Substitution
Input Rows : 3
1.Encrypt
2.Decrypt
Enter your choice: 1
P h t a o b u e l t t i y c i S a u o b l s n t p i
```

RESULT:

Thus, a python program has been implemented to demonstrate Rail Fence Cipher.

EXP NO: 1d

DATE:

## COLUMNAR TRANSPOSITION TECHNIQUES

AIM:

To write a python program to implement columnar transposition techniques.

ALGORITHM:

1. Select a keyword that will determine the order of columns.
2. If the keyword has duplicate letters, remove the duplicates, keeping only the first occurrence of each letter.
3. Assign a numerical position to each letter in the keyword based on alphabetical order. For example, if the keyword is "CIPHER", then the numerical positions .
4. Write the plaintext message in rows, using a grid with as many columns as there are unique letters in the keyword.
5. If the last row is not completely filled, add padding characters (commonly 'X' or any other character) to fill the grid.
6. Read the columns of the grid in the numerical order of the keyword positions.
7. Combine the characters read in the order of the columns to form the ciphertext.
8. To decrypt, create an empty grid with the same number of columns as the keyword.

## PROGRAM:

```
import math

key = input("Enter the key ")

# Encryption
def encryptMessage(msg):
    cipher = ""

    # track key indices
    k_indx = 0

    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))

    # calculate column of the matrix
    col = len(key)

    # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))

    # add the padding character '_' in empty
    # the empty cell of the matrix
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)

    # create Matrix and insert message and
    # padding characters row-wise
    matrix = [msg_lst[i: i + col]
               for i in range(0, len(msg_lst), col)]

    # read matrix column-wise using key
    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ".join([row[curr_idx]
```

```

                                for row in matrix])

        k_indx += 1

    return cipher

# Decryption
def decryptMessage(cipher):
    msg = ""

    # track key indices
    k_indx = 0

    # track msg indices
    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)

    # calculate column of the matrix
    col = len(key)

    # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))

    # convert key into list and sort
    # alphabetically so we can access
    # each character by its alphabetical position.
    key_lst = sorted(list(key))

    # create an empty matrix to
    # store deciphered message
    dec_cipher = []
    for _ in range(row):
        dec_cipher += [[None] * col]

    # Arrange the matrix column wise according
    # to permutation order by adding into new matrix
    for _ in range(col):

```



```

curr_idx = key.index(key_lst[k_idx])

for j in range(row):
    dec_cipher[j][curr_idx] = msg_lst[msg_idx]
    msg_idx += 1
    k_idx += 1

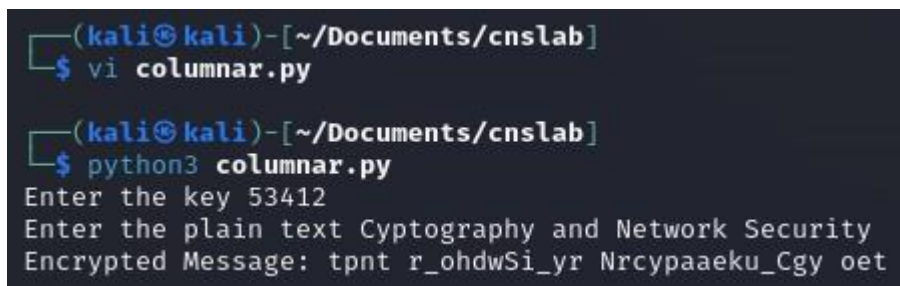
# convert decrypted msg matrix into a string
try:
    msg = ''.join(sum(dec_cipher, []))
except TypeError:
    raise TypeError("This program cannot", "handle repeating words.")
null_count = msg.count('_')
if null_count > 0:
    return msg[: -null_count]
return msg

msg = input("Enter the plain text ")
cipher = encryptMessage(msg)
print("Encrypted Message: {}".format(cipher))

print("Decrypted Message: {}".format(decryptMessage(cipher)))

```

OUTPUT:



```

(kali@kali)-[~/Documents/cnslab]
$ vi columnar.py

(kali@kali)-[~/Documents/cnslab]
$ python3 columnar.py
Enter the key 53412
Enter the plain text Cyptography and Network Security
Encrypted Message: tpnt r_ohdwSi_yr Nrcypaaeku_Cgy oet

```

RESULT:

Thus, a python program has been implemented to demonstrate Columnar Transposition techniques.