1.Write a C program for Caesar cipher involves replacing each letter of the alphabet with the letter standing k places further down the alphabet, for k in the range 1 through 25.

Program:

def caesar\_cipher(text, shift):

result = ""

for char in text:

if char.isalpha():

offset = 65 if char.isupper() else 97

shifted\_char = chr((ord(char) - offset + shift) % 26 + offset)

result += shifted\_char

else:

result += char

return result

text = input("Enter the text: ")

shift = int(input("Enter the shift value (1-25): "))

if 1 <= shift <= 25:

encrypted\_text = caesar\_cipher(text, shift)

decrypted\_text = caesar\_cipher(encrypted\_text, -shift)

print(f"Encrypted text: {encrypted\_text}")

print(f"Decrypted text: {decrypted\_text}")

else:

print("Shift value must be between 1 and 25.")

Output:

Enter the text: harshini

Enter the shift value (1-25): 5

Encrypted text: mfwxmnsn

Decrypted text: Harshini

2. Write a C program for monoalphabetic substitution cipher maps a plaintext alphabet to a ciphertext alphabet, so that each letter of the plaintext alphabet maps to a single unique letter of the ciphertext alphabet.

Program:

def mono\_substitution\_cipher(text, key):

cipher = str.maketrans('abcdefghijklmnopqrstuvwxyz', key)

return text.translate(cipher)

key = 'zyxwvutsrqponmlkjihgfedcba'

plaintext = input("Enter the text to encrypt: ")

encrypted\_text = mono\_substitution\_cipher(plaintext, key)

print(f"Encrypted text: {encrypted\_text}")

Output:

Enter the text to encrypt: harshini

Encrypted text: szihsrmr

3. Write a C program for Playfair algorithm is based on the use of a 5 X 5 matrix of letters constructed using a keyword. Plaintext is encrypted two letters at a time using this matrix.

Program:

def create\_matrix(key):

key = key.upper().replace("J", "I") + "ABCDEFGHIKLMNOPQRSTUVWXYZ"

matrix = []

for char in key:

if char not in matrix:

matrix.append(char)

return matrix

def playfair\_cipher(key, text):

matrix = create\_matrix(key)

text = text.upper().replace("J", "I").replace(" ", "")

cipher = []

for i in range(0, len(text), 2):

c1, c2 = text[i], text[i + 1] if i + 1 < len(text) else 'X'

r1, c1 = divmod(matrix.index(c1), 5)

r2, c2 = divmod(matrix.index(c2), 5)

if r1 == r2:

cipher.append(matrix[r1 \* 5 + (c1 + 1) % 5])

cipher.append(matrix[r2 \* 5 + (c2 + 1) % 5])

elif c1 == c2:

cipher.append(matrix[(r1 + 1) % 5 \* 5 + c1])

cipher.append(matrix[(r2 + 1) % 5 \* 5 + c2])

else:

cipher.append(matrix[r1 \* 5 + c2])

cipher.append(matrix[r2 \* 5 + c1])

return ''.join(cipher)

key = input("Keyword: ")

text = input("Text: ")

print("Cipher: " + playfair\_cipher(key, text))

Output:

Keyword: hello

Text: HAI

Cipher: EHMV

4. Write a C program for polyalphabetic substitution cipher uses a separate monoalphabetic substitution cipher for each successive letter of plaintext, depending on a key.

Program:

def polyalphabetic\_cipher(text, key):

encrypted\_text = ""

key\_length = len(key)

for i in range(len(text)):

char = text[i]

if char.isalpha():

shift = ord(key[i % key\_length].lower()) - ord('a')

if char.isupper():

encrypted\_char = chr(((ord(char) - ord('A') + shift) % 26) + ord('A'))

else:

encrypted\_char = chr(((ord(char) - ord('a') + shift) % 26) + ord('a'))

else:

encrypted\_char = char

encrypted\_text += encrypted\_char

return encrypted\_text

message = input("Enter a message: ")

keyword = input("Enter a keyword: ")

encrypted\_message = polyalphabetic\_cipher(message, keyword)

print("Encrypted message:", encrypted\_message)

Output:

Enter a message: hello

Enter a keyword: 1234

Encrypted message: ljrss

5. Write a C program forgeneralization of the Caesar cipher, known as the affine Caesar cipher, has thefollowing form: For each plaintext letter p, substitute the ciphertextletterC: C = E([a, b], p) = (ap + b) mod 26 A basic requirement of any encryption algorithm is that it be one-to-one. That is, if p q,then E(k, p) E(k, q). Otherwise, decryption is impossible, because more than one plaintext character maps into the same ciphertext character. The affine Caesar cipher is not one-to one for all values of a. For example, for a = 2 and b = 3,then E([a,b], 0) = E([a, b], 13) = 3.

a. Are there any limitations on the value of b?

b. Determine which values of a are not allowed.

Program:

def affine\_caesar(text, a, b, encrypt=True):

result = ""

for char in text:

if char.isalpha():

shift = ord('A') if char.isupper() else ord('a')

char\_num = ord(char) - shift

if encrypt:

new\_char\_num = (a \* char\_num + b) % 26

else:

a\_inverse = None

for i in range(26):

if (a \* i) % 26 == 1:

a\_inverse = i

break

if a\_inverse is None:

return "Decryption not possible with this 'a' value."

new\_char\_num = (a\_inverse \* (char\_num - b)) % 26

result += chr(new\_char\_num + shift)

else:

result += char

return result

plaintext = input("Enter the text: ")

a = int(input("Enter the A value: "))

b = int(input("Enter the B value: "))

encrypted\_text = affine\_caesar(plaintext, a, b, encrypt=True)

print("Encrypted:", encrypted\_text)

decrypted\_text = affine\_caesar(encrypted\_text, a, b, encrypt=False)

print("Decrypted:", decrypted\_text)

Output:

Enter the text: hello

Enter the A value: 15

Enter the B value: 5

Encrypted: gnooh

Decrypted: hello

6. Write a C program for cipher text has been generated with an affine cipher most frequent letter of the ciphertext is "B," and the second most frequent letter of the ciphertext is "U." Break this code.

Program:

def break\_affine\_cipher(c, a, b):

return ''.join([chr((a \* (ord(x) - 65 - b) % 26) + 97) if x.isalpha() else x for x in c])

ciphertext = input("Enter the ciphertext: ").upper()

most\_common = 'B'

second\_most\_common = 'U'

a = (ord(most\_common) - ord(second\_most\_common)) % 26

b = (ord(most\_common) - ord('A')) % 26

plaintext = break\_affine\_cipher(ciphertext, a, b)

print(plaintext)

Output:

Enter the ciphertext: hello

qvssn