**PRACTICAL-5**

**AIM: Write a C program for encryption and decryption of Hill Cipher.**

**INRODUCTION:**

* Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.
* The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

**CODE:**

#include<stdio.h>

#include<math.h>

float encrypt[3][1], decrypt[3][1], a[3][3], b[3][3], mes[3][1], c[3][3];

void encryption(); //encrypts the message

void decryption(); //decrypts the message

void getKeyMessage(); //gets key and message from user

void inverse(); //finds inverse of key matrix

int main() {

getKeyMessage();

encryption();

decryption(); }

void encryption() {

int i, j, k;

for(i = 0; i < 3; i++)

for(j = 0; j < 1; j++)

for(k = 0; k < 3; k++)

encrypt[i][j] = encrypt[i][j] + a[i][k] \* mes[k][j];

printf("\nEncrypted string is: ");

for(i = 0; i < 3; i++)

printf("%c", (char)(fmod(encrypt[i][0], 26) + 97)); }

void decryption() {

int i, j, k;

inverse();

for(i = 0; i < 3; i++)

for(j = 0; j < 1; j++)

for(k = 0; k < 3; k++)

decrypt[i][j] = decrypt[i][j] + b[i][k] \* encrypt[k][j];

printf("\nDecrypted string is: ");

for(i = 0; i < 3; i++)

printf("%c", (char)(fmod(decrypt[i][0], 26) + 97));

printf("\n");

}

void getKeyMessage() {

int i, j;

char msg[3];

printf("Enter 3x3 matrix for key (It should be inversible):\n");

for(i = 0; i < 3; i++)

for(j = 0; j < 3; j++) {

scanf("%f", &a[i][j]);

c[i][j] = a[i][j];

}

printf("\nEnter a 3 letter string: ");

scanf("%s", msg);

for(i = 0; i < 3; i++)

mes[i][0] = msg[i] - 97;

}

void inverse() {

int i, j, k;

float p, q;

for(i = 0; i < 3; i++)

for(j = 0; j < 3; j++) {

if(i == j)

b[i][j]=1;

else

b[i][j]=0; }

for(k = 0; k < 3; k++) {

for(i = 0; i < 3; i++) {

p = c[i][k];

q = c[k][k];

for(j = 0; j < 3; j++) {

if(i != k) {

c[i][j] = c[i][j]\*q - p\*c[k][j];

b[i][j] = b[i][j]\*q - p\*b[k][j];

} } } }

for(i = 0; i < 3; i++)

for(j = 0; j < 3; j++)

b[i][j] = b[i][j] / c[i][i];

printf("\n\nInverse Matrix is:\n");

for(i = 0; i < 3; i++) {

for(j = 0; j < 3; j++)

printf("%d ", b[i][j]);

printf("\n"); } }

**OUTPUT**:

