# **LAB 1**

1. WAP in C implement Ceasar Cipher.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

// Function to encrypt the text using Caesar Cipher

void encrypt(char text[], int shift) {

for (int i = 0; text[i] != '\0'; i++) {

char ch = text[i];

if (isalpha(ch)) {

char base = islower(ch) ? 'a' : 'A';

text[i] = (ch - base + shift) % 26 + base;

}

}

}

// Function to decrypt the text using Caesar Cipher

void decrypt(char text[], int shift) {

encrypt(text, 26 - (shift % 26)); // Decrypt by reversing the shift

}

int main() {

char text[100];

int shift, choice;

printf("Enter the text: ");

fgets(text, sizeof(text), stdin);

text[strcspn(text, "\n")] = '\0'; // Remove newline character from input

printf("Enter the shift value: ");

scanf("%d", &shift);

printf("Choose an option:\n1. Encrypt\n2. Decrypt\nYour choice: ");

scanf("%d", &choice);

if (choice == 1) {

encrypt(text, shift);

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

} else if (choice == 2) {

decrypt(text, shift);

printf("Decrypted text: %s\n", text);

} else {

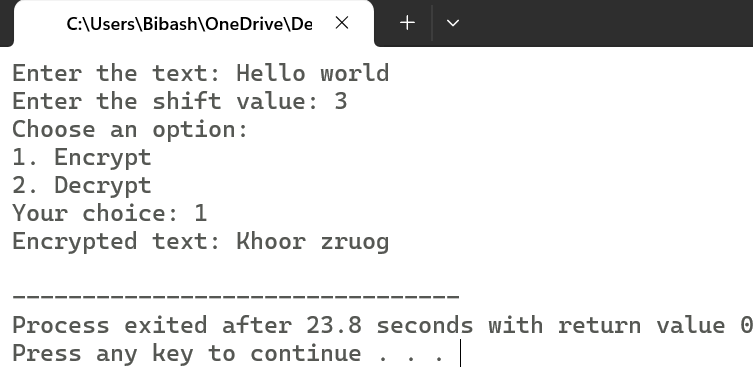
printf("Invalid choice.\n");

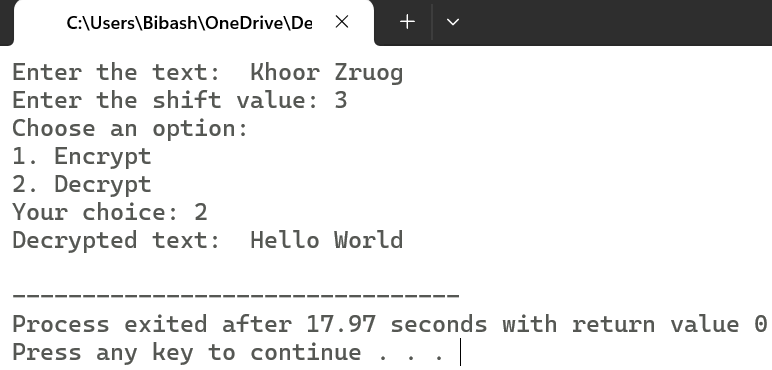
}

return 0;

}

OUTPUT





1. WAP in C implement Hill Cipher.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

// Function to compute the determinant of a 2x2 matrix

int determinant(int key[2][2]) {

return (key[0][0] \* key[1][1] - key[0][1] \* key[1][0]);

}

// Function to calculate modular inverse

int modInverse(int a, int m) {

a = a % m;

for (int x = 1; x < m; x++) {

if ((a \* x) % m == 1)

return x;

}

return -1; // Modular inverse does not exist

}

// Function to prepare the plaintext

void prepareText(char text[]) {

int len = strlen(text);

if (len % 2 != 0) {

text[len] = 'x'; // Append 'x' if length is odd

text[len + 1] = '\0';

}

}

// Function to encrypt plaintext using Hill cipher

void encrypt(char plaintext[], int key[2][2], char ciphertext[]) {

int len = strlen(plaintext);

for (int i = 0; i < len; i += 2) {

int p1 = plaintext[i] - 'a';

int p2 = plaintext[i + 1] - 'a';

// Apply the key matrix

ciphertext[i] = ((key[0][0] \* p1 + key[0][1] \* p2) % 26) + 'a';

ciphertext[i + 1] = ((key[1][0] \* p1 + key[1][1] \* p2) % 26) + 'a';

}

ciphertext[len] = '\0'; // Null-terminate the ciphertext

}

// Function to decrypt ciphertext using Hill cipher

void decrypt(char ciphertext[], int key[2][2], char plaintext[]) {

int det = determinant(key);

int detMod = (det % 26 + 26) % 26; // Determinant mod 26

int invDet = modInverse(detMod, 26);

if (invDet == -1) {

printf("Matrix is not invertible under mod 26.\n");

return;

}

// Compute the inverse key matrix

int invKey[2][2];

invKey[0][0] = (key[1][1] \* invDet) % 26;

invKey[0][1] = (-key[0][1] \* invDet + 26) % 26;

invKey[1][0] = (-key[1][0] \* invDet + 26) % 26;

invKey[1][1] = (key[0][0] \* invDet) % 26;

int len = strlen(ciphertext);

for (int i = 0; i < len; i += 2) {

int c1 = ciphertext[i] - 'a';

int c2 = ciphertext[i + 1] - 'a';

// Apply the inverse key matrix

plaintext[i] = ((invKey[0][0] \* c1 + invKey[0][1] \* c2) % 26) + 'a';

plaintext[i + 1] = ((invKey[1][0] \* c1 + invKey[1][1] \* c2) % 26) + 'a';

}

plaintext[len] = '\0'; // Null-terminate the plaintext

}

int main() {

int key[2][2];

char plaintext[100], ciphertext[100];

int choice;

printf("Enter 2x2 key matrix (integers):\n");

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

scanf("%d", &key[i][j]);

}

}

printf("Enter the text: ");

scanf("%s", plaintext);

// Convert text to lowercase and remove non-alphabet characters

for (int i = 0; plaintext[i]; i++) {

plaintext[i] = tolower(plaintext[i]);

}

prepareText(plaintext);

printf("Choose an option:\n1. Encrypt\n2. Decrypt\nYour choice: ");

scanf("%d", &choice);

if (choice == 1) {

encrypt(plaintext, key, ciphertext);

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

} else if (choice == 2) {

decrypt(plaintext, key, ciphertext);

printf("Decrypted text: %s\n", ciphertext);

} else {

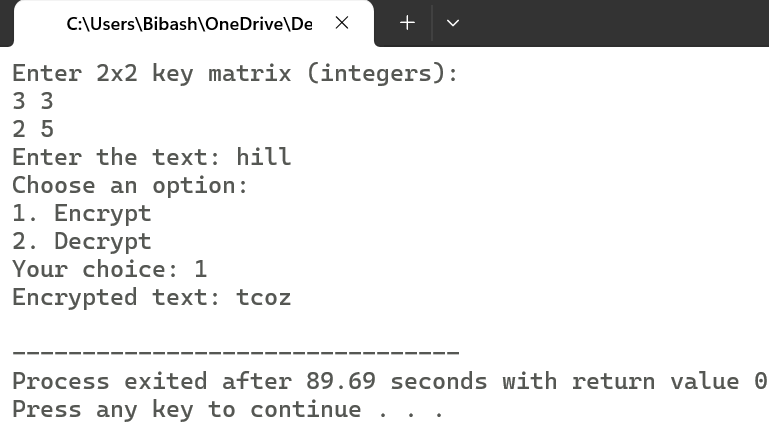
printf("Invalid choice.\n");

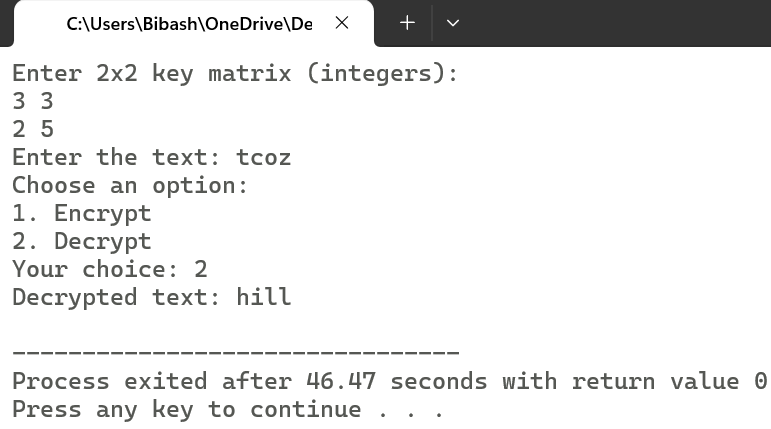
}

return 0;

}

OUTPUT:





1. WAP in C implement Play Fair Cipher.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define SIZE 5

// Function to create the Playfair cipher key matrix

void createMatrix(char key[], char matrix[SIZE][SIZE]) {

int alpha[26] = {0};

alpha['j' - 'a'] = 1; // Treat 'j' as 'i'

int keyLen = strlen(key), k = 0;

// Fill the matrix with the keyword

for (int i = 0; i < keyLen; i++) {

char ch = tolower(key[i]);

if (isalpha(ch) && !alpha[ch - 'a']) {

alpha[ch - 'a'] = 1;

matrix[k / SIZE][k % SIZE] = ch;

k++;

}

}

// Fill the rest of the matrix with remaining letters

for (char ch = 'a'; ch <= 'z'; ch++) {

if (!alpha[ch - 'a']) {

matrix[k / SIZE][k % SIZE] = ch;

k++;

}

if (ch == 'i') ch++; // Skip 'j'

}

}

// Function to display the cipher matrix

void displayMatrix(char matrix[SIZE][SIZE]) {

printf("Playfair Matrix:\n");

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

printf("%c ", matrix[i][j]);

}

printf("\n");

}

}

// Function to find the position of a character in the matrix

void findPosition(char matrix[SIZE][SIZE], char ch, int \*row, int \*col) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (matrix[i][j] == ch) {

\*row = i;

\*col = j;

return;

}

}

}

}

// Function to prepare the plaintext (convert to pairs)

void prepareText(char text[]) {

int len = strlen(text), k = 0;

char prepared[100];

for (int i = 0; i < len; i++) {

if (isalpha(text[i])) {

prepared[k++] = tolower(text[i]);

if (k > 1 && prepared[k - 1] == prepared[k - 2]) {

prepared[k - 1] = 'x'; // Insert 'x' if repeated character in a pair

i--; // Check the same character again

}

}

}

if (k % 2 != 0) prepared[k++] = 'x'; // Append 'x' if odd length

prepared[k] = '\0';

strcpy(text, prepared);

}

// Function to encrypt/decrypt text using Playfair Cipher

void processText(char text[], char matrix[SIZE][SIZE], int encrypt) {

int len = strlen(text);

for (int i = 0; i < len; i += 2) {

int r1, c1, r2, c2;

findPosition(matrix, text[i], &r1, &c1);

findPosition(matrix, text[i + 1], &r2, &c2);

if (r1 == r2) { // Same row

text[i] = matrix[r1][(c1 + encrypt + SIZE) % SIZE];

text[i + 1] = matrix[r2][(c2 + encrypt + SIZE) % SIZE];

} else if (c1 == c2) { // Same column

text[i] = matrix[(r1 + encrypt + SIZE) % SIZE][c1];

text[i + 1] = matrix[(r2 + encrypt + SIZE) % SIZE][c2];

} else { // Rectangle swap

text[i] = matrix[r1][c2];

text[i + 1] = matrix[r2][c1];

}

}

}

int main() {

char key[100], text[100];

char matrix[SIZE][SIZE];

int choice;

printf("Enter the keyword: ");

fgets(key, sizeof(key), stdin);

key[strcspn(key, "\n")] = '\0';

createMatrix(key, matrix);

displayMatrix(matrix);

printf("Enter the text: ");

fgets(text, sizeof(text), stdin);

text[strcspn(text, "\n")] = '\0';

prepareText(text);

printf("Prepared text: %s\n", text);

printf("Choose an option:\n1. Encrypt\n2. Decrypt\nYour choice: ");

scanf("%d", &choice);

if (choice == 1) {

processText(text, matrix, 1);

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

} else if (choice == 2) {

processText(text, matrix, -1);

printf("Decrypted text: %s\n", text);

} else {

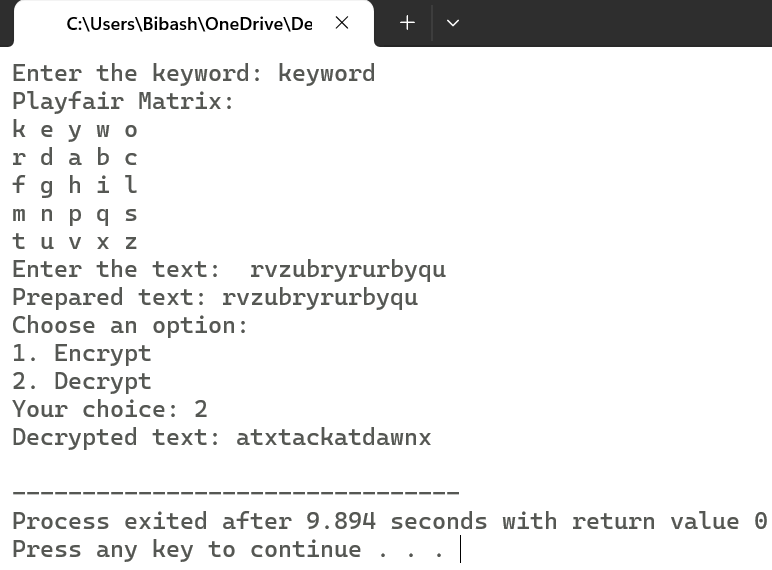
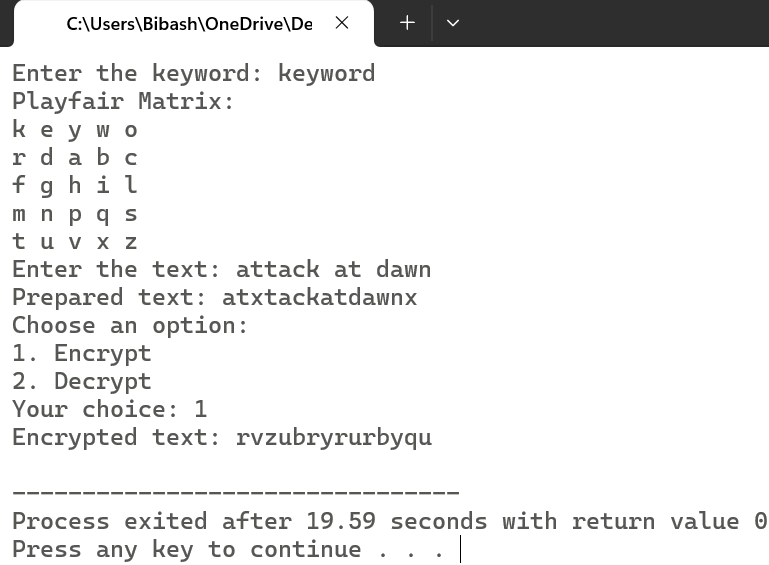
printf("Invalid choice.\n");

}

return 0;

}

OUTPUT:



1. WAP in C implement Polyalphabetic Cipher.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

// Function to encrypt the text using Polyalphabetic Cipher

void encrypt(char text[], char key[]) {

int textLen = strlen(text);

int keyLen = strlen(key);

char encryptedText[100];

for (int i = 0, j = 0; i < textLen; i++) {

char ch = text[i];

if (isalpha(ch)) {

char base = islower(ch) ? 'a' : 'A';

char shift = tolower(key[j % keyLen]) - 'a'; // Normalize key character to a shift value

encryptedText[i] = (ch - base + shift) % 26 + base;

j++; // Increment key index only for alphabetic characters

} else {

encryptedText[i] = ch; // Non-alphabet characters remain unchanged

}

}

encryptedText[textLen] = '\0'; // Null-terminate the string

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

}

// Function to decrypt the text using Polyalphabetic Cipher

void decrypt(char text[], char key[]) {

int textLen = strlen(text);

int keyLen = strlen(key);

char decryptedText[100];

for (int i = 0, j = 0; i < textLen; i++) {

char ch = text[i];

if (isalpha(ch)) {

char base = islower(ch) ? 'a' : 'A';

char shift = tolower(key[j % keyLen]) - 'a'; // Normalize key character to a shift value

decryptedText[i] = (ch - base - shift + 26) % 26 + base;

j++; // Increment key index only for alphabetic characters

} else {

decryptedText[i] = ch; // Non-alphabet characters remain unchanged

}

}

decryptedText[textLen] = '\0'; // Null-terminate the string

printf("Decrypted text: %s\n", decryptedText);

}

int main() {

char text[100], key[100];

int choice;

printf("Enter the text: ");

fgets(text, sizeof(text), stdin);

text[strcspn(text, "\n")] = '\0'; // Remove newline character from input

printf("Enter the keyword: ");

fgets(key, sizeof(key), stdin);

key[strcspn(key, "\n")] = '\0'; // Remove newline character from input

printf("Choose an option:\n1. Encrypt\n2. Decrypt\nYour choice: ");

scanf("%d", &choice);

if (choice == 1) {

encrypt(text, key);

} else if (choice == 2) {

decrypt(text, key);

} else {

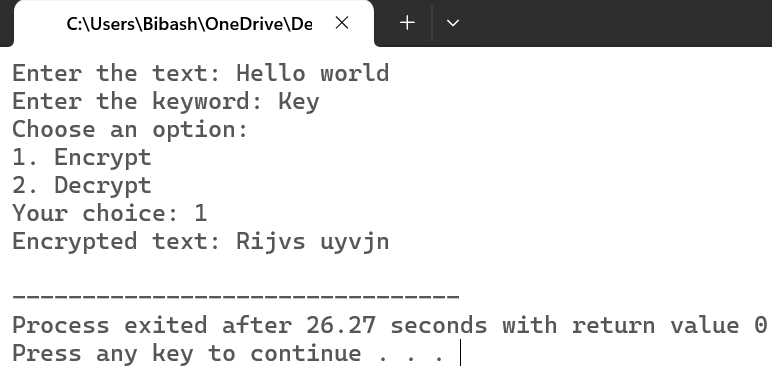
printf("Invalid choice.\n");

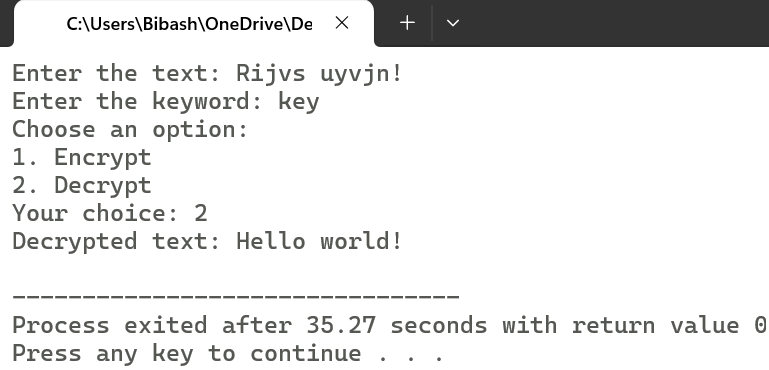
}

return 0;

}

OUTPUT:





# LAB 2

1. WAP in C implement Row Transpose Cipher.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

// Function to encrypt using Row Transposition Cipher

void encrypt(char \*plaintext, int key[], int key\_length) {

int len = strlen(plaintext);

int rows = (len + key\_length - 1) / key\_length; // Calculate rows needed

char matrix[rows][key\_length];

// Fill the matrix with plaintext characters

int k = 0;

for (int i = 0; i < rows; i++) {

for (int j = 0; j < key\_length; j++) {

if (k < len)

matrix[i][j] = plaintext[k++];

else

matrix[i][j] = 'X'; // Padding with 'X' if needed

}

}

// Print ciphertext column-wise according to key order

printf("Encrypted text: ");

for (int i = 0; i < key\_length; i++) {

int col = key[i] - 1;

for (int j = 0; j < rows; j++) {

printf("%c", matrix[j][col]);

}

}

printf("\n");

}

// Function to decrypt Row Transposition Cipher

void decrypt(char \*ciphertext, int key[], int key\_length) {

int len = strlen(ciphertext);

int rows = (len + key\_length - 1) / key\_length;

char matrix[rows][key\_length];

int k = 0;

// Fill the matrix column-wise according to key order

for (int i = 0; i < key\_length; i++) {

int col = key[i] - 1;

for (int j = 0; j < rows; j++) {

if (k < len)

matrix[j][col] = ciphertext[k++];

}

}

// Read the matrix row-wise to get plaintext

printf("Decrypted text: ");

for (int i = 0; i < rows; i++) {

for (int j = 0; j < key\_length; j++) {

if (matrix[i][j] != 'X') // Avoid printing padding characters

printf("%c", matrix[i][j]);

}

}

printf("\n");

}

int main() {

char plaintext[100], ciphertext[100];

int key[] = {3, 1, 4, 2, 5}; // Key permutation (change as needed)

int key\_length = sizeof(key) / sizeof(key[0]);

printf("Enter plaintext: ");

fgets(plaintext, sizeof(plaintext), stdin);

plaintext[strcspn(plaintext, "\n")] = 0; // Remove newline character

encrypt(plaintext, key, key\_length);

printf("Enter ciphertext to decrypt: ");

fgets(ciphertext, sizeof(ciphertext), stdin);

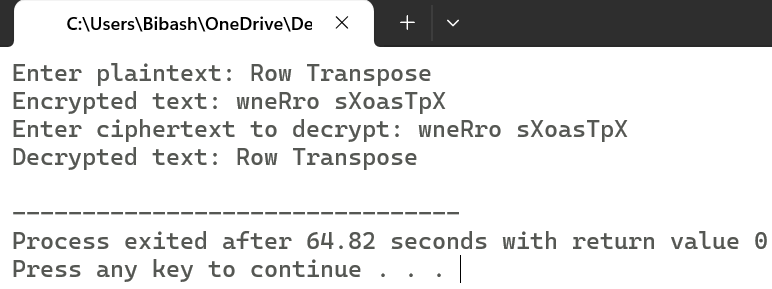
ciphertext[strcspn(ciphertext, "\n")] = 0; // Remove newline character

decrypt(ciphertext, key, key\_length);

return 0;

}

OUTPUT:



1. WAP in C implement Rail-Fence Cipher.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

// Function to encrypt using Rail Fence Cipher

void encrypt(char \*plaintext, int rails) {

int len = strlen(plaintext);

char rail[rails][len];

// Initialize the rail matrix with 'nul’

for (int i = 0; i < rails; i++)

for (int j = 0; j < len; j++)

rail[i][j] = '\0';

// Fill the rail matrix in a zigzag pattern

int row = 0, direction = 1;

for (int i = 0; i < len; i++) {

rail[row][i] = plaintext[i];

if (row == 0)

direction = 1;

else if (row == rails - 1)

direction = -1;

row += direction;

}

// Read the matrix row-wise to get the ciphertext

printf("Encrypted text: ");

for (int i = 0; i < rails; i++)

for (int j = 0; j < len; j++)

if (rail[i][j] != '\0')

printf("%c", rail[i][j]);

printf("\n");

}

// Function to decrypt Rail Fence Cipher

void decrypt(char \*ciphertext, int rails) {

int len = strlen(ciphertext);

char rail[rails][len];

// Initialize the rail matrix with '\*'

for (int i = 0; i < rails; i++)

for (int j = 0; j < len; j++)

rail[i][j] = '\*';

// Mark positions in a zigzag pattern

int row = 0, direction = 1;

for (int i = 0; i < len; i++) {

rail[row][i] = 'X';

if (row == 0)

direction = 1;

else if (row == rails - 1)

direction = -1;

row += direction;

}

// Fill the marked positions with ciphertext characters

int index = 0;

for (int i = 0; i < rails; i++)

for (int j = 0; j < len; j++)

if (rail[i][j] == 'X' && index < len)

rail[i][j] = ciphertext[index++];

// Read the matrix in a zigzag pattern to reconstruct plaintext

row = 0, direction = 1;

printf("Decrypted text: ");

for (int i = 0; i < len; i++) {

printf("%c", rail[row][i]);

if (row == 0)

direction = 1;

else if (row == rails - 1)

direction = -1;

row += direction;

}

printf("\n");

}

int main() {

char plaintext[100], ciphertext[100];

int rails;

printf("Enter number of rails: ");

scanf("%d", &rails);

getchar(); // Consume newline character

printf("Enter plaintext: ");

fgets(plaintext, sizeof(plaintext), stdin);

plaintext[strcspn(plaintext, "\n")] = 0;

encrypt(plaintext, rails);

printf("Enter ciphertext to decrypt: ");

fgets(ciphertext, sizeof(ciphertext), stdin);

ciphertext[strcspn(ciphertext, "\n")] = 0;

decrypt(ciphertext, rails);

return 0;

}

OUTPUT:

