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.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define SIZE 5

// Function to convert the input to uppercase and replace J with I

void preprocessInput(char \*input) {

int i;

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

input[i] = toupper(input[i]);

if (input[i] == 'J')

input[i] = 'I';

}

}

// Function to create the Playfair matrix using a keyword

void createPlayfairMatrix(char matrix[SIZE][SIZE], const char \*keyword) {

char key[26] = {0};

int i, j, k = 0;

int used[26] = {0};

// Preprocess the keyword: convert to uppercase and replace J with I

char processed\_keyword[100];

strcpy(processed\_keyword, keyword);

preprocessInput(processed\_keyword);

// First, fill in the keyword (avoiding duplicates)

for (i = 0; processed\_keyword[i] != '\0'; i++) {

if (isalpha(processed\_keyword[i])) {

int index = processed\_keyword[i] - 'A';

if (!used[index]) {

key[k++] = processed\_keyword[i];

used[index] = 1;

}

}

}

// Then fill in the remaining letters (except J)

for (i = 0; i < 26; i++) {

if (i != ('J' - 'A') && !used[i]) {

key[k++] = 'A' + i;

}

}

// Fill the matrix

k = 0;

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

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

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

}

}

}

// Function to display the Playfair matrix

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

int i, j;

printf("\nPlayfair Matrix (5x5):\n");

printf(" | 1 | 2 | 3 | 4 | 5 |\n");

printf("--+---+---+---+---+---+\n");

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

printf("%d | ", i + 1);

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

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

}

printf("\n");

}

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) {

int i, j;

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

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

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

\*row = i;

\*col = j;

return;

}

}

}

}

// Function to prepare the plaintext for encryption

void prepareText(char \*text) {

int i, j = 0;

char prepared[1000] = {0};

// Remove non-alphabetic characters and convert to uppercase

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

if (isalpha(text[i])) {

prepared[j++] = toupper(text[i]);

if (prepared[j-1] == 'J')

prepared[j-1] = 'I';

}

}

prepared[j] = '\0';

// Insert 'X' between double letters

j = 0;

for (i = 0; i < strlen(prepared); i++) {

text[j++] = prepared[i];

if (i < strlen(prepared) - 1 && prepared[i] == prepared[i+1]) {

text[j++] = 'X';

}

}

// Add 'X' if the length is odd

if (j % 2 != 0) {

text[j++] = 'X';

}

text[j] = '\0';

}

// Function to encrypt a message using the Playfair cipher

void encrypt(char matrix[SIZE][SIZE], const char \*input, char \*output) {

int i;

int row1, col1, row2, col2;

for (i = 0; i < strlen(input); i += 2) {

findPosition(matrix, input[i], &row1, &col1);

findPosition(matrix, input[i+1], &row2, &col2);

// Same row: take character to the right (wrapping around)

if (row1 == row2) {

output[i] = matrix[row1][(col1 + 1) % SIZE];

output[i+1] = matrix[row2][(col2 + 1) % SIZE];

}

// Same column: take character below (wrapping around)

else if (col1 == col2) {

output[i] = matrix[(row1 + 1) % SIZE][col1];

output[i+1] = matrix[(row2 + 1) % SIZE][col2];

}

// Rectangle: take character in the same row but in the column of the other character

else {

output[i] = matrix[row1][col2];

output[i+1] = matrix[row2][col1];

}

}

output[i] = '\0';

}

// Function to decrypt a message using the Playfair cipher

void decrypt(char matrix[SIZE][SIZE], const char \*input, char \*output) {

int i;

int row1, col1, row2, col2;

for (i = 0; i < strlen(input); i += 2) {

findPosition(matrix, input[i], &row1, &col1);

findPosition(matrix, input[i+1], &row2, &col2);

// Same row: take character to the left (wrapping around)

if (row1 == row2) {

output[i] = matrix[row1][(col1 - 1 + SIZE) % SIZE];

output[i+1] = matrix[row2][(col2 - 1 + SIZE) % SIZE];

}

// Same column: take character above (wrapping around)

else if (col1 == col2) {

output[i] = matrix[(row1 - 1 + SIZE) % SIZE][col1];

output[i+1] = matrix[(row2 - 1 + SIZE) % SIZE][col2];

}

// Rectangle: take character in the same row but in the column of the other character

else {

output[i] = matrix[row1][col2];

output[i+1] = matrix[row2][col1];

}

}

output[i] = '\0';

}

// Function to format and display digraphs

void displayDigraphs(const char \*text) {

int i;

printf("Digraphs: ");

for (i = 0; i < strlen(text); i += 2) {

printf("%c%c ", text[i], text[i+1]);

}

printf("\n\n");

}

int main() {

char matrix[SIZE][SIZE];

char keyword[100];

char plaintext[1000], prepared\_text[1000];

char ciphertext[1000], decrypted[1000];

int choice;

printf("===== PLAYFAIR CIPHER PROGRAM =====\n\n");

printf("Enter the keyword: ");

fgets(keyword, sizeof(keyword), stdin);

// Remove newline character from keyword

if (keyword[strlen(keyword) - 1] == '\n')

keyword[strlen(keyword) - 1] = '\0';

// Create the Playfair matrix using the keyword

createPlayfairMatrix(matrix, keyword);

// Display the matrix

displayMatrix(matrix);

printf("Choose an operation:\n");

printf("1. Encrypt\n");

printf("2. Decrypt\n");

printf("Enter your choice (1/2): ");

scanf("%d", &choice);

getchar(); // Consume the newline

if (choice == 1) {

// Encryption

printf("Enter the plaintext: ");

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

// Remove newline character from plaintext

if (plaintext[strlen(plaintext) - 1] == '\n')

plaintext[strlen(plaintext) - 1] = '\0';

// Prepare the plaintext for Playfair encryption

strcpy(prepared\_text, plaintext);

prepareText(prepared\_text);

printf("\nPrepared text: %s\n", prepared\_text);

displayDigraphs(prepared\_text);

// Encrypt the prepared plaintext

encrypt(matrix, prepared\_text, ciphertext);

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

displayDigraphs(ciphertext);

} else if (choice == 2) {

// Decryption

printf("Enter the ciphertext (uppercase, no spaces): ");

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

// Remove newline character from ciphertext

if (ciphertext[strlen(ciphertext) - 1] == '\n')

ciphertext[strlen(ciphertext) - 1] = '\0';

// Convert to uppercase

preprocessInput(ciphertext);

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

displayDigraphs(ciphertext);

// Decrypt the ciphertext

decrypt(matrix, ciphertext, decrypted);

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

displayDigraphs(decrypted);

printf("Note: The decrypted text may contain 'X' characters that were added during encryption.\n");

} else {

printf("Invalid choice. Please run the program again.\n");

return 1;

}

return 0;

}

Output:

