DAY 2

1. Write a High level codefor monoalphabetic cipher is that both sender and receiver must commit the

permuted cipher sequence to memory. A common technique for avoiding this is to use a keyword from

which the cipher sequence can be generated.

For example, using the keyword CIPHER, write out the keyword followed by unused letters in normal

order and match this against the plaintext letters:

PROGRAM

#include <stdio.h>

#include <string.h>

int main() {

char plaintext[] = "abcdefghijklmnopqrstuvwxyz"; // Plaintext alphabet

char ciphertext[] = "zyxwvutsrqponmlkjihgfedcba"; // Ciphertext alphabet

char input[100], output[100];

printf("Enter a message to encrypt: ");

fgets(input, sizeof(input), stdin);

int length = strlen(input);

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

if (input[i] >= 'a' && input[i] <= 'z') {

int index = input[i] - 'a';

output[i] = ciphertext[index];

} else {

output[i] = input[i]; // Keep non-alphabet characters as is

}

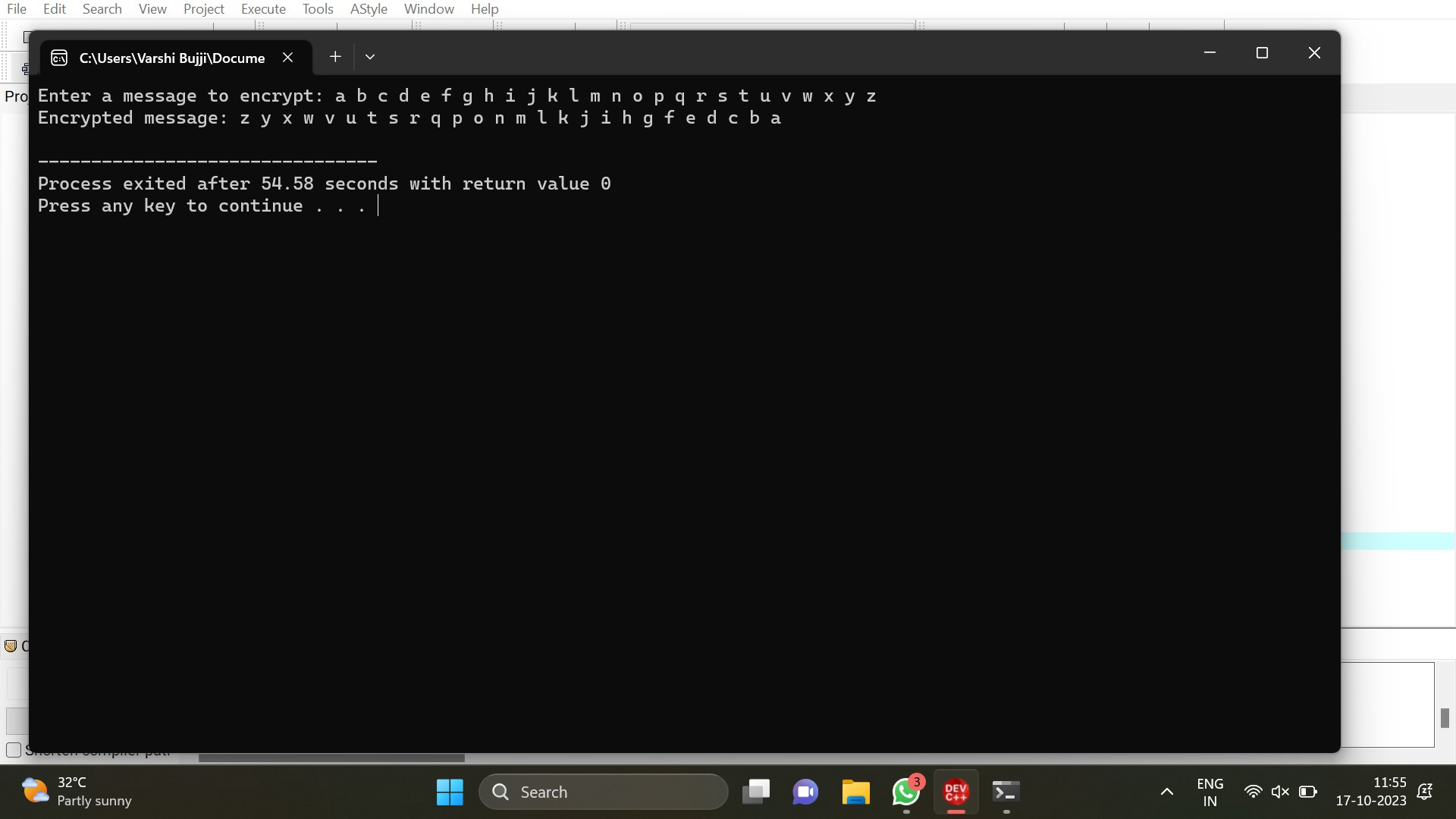
}

printf("Encrypted message: %s", output);

return 0;

}

OUTPUT



2. Write a High level codefor PT-109 American patrol boat, under the command of Lieutenant John F.

Kennedy, was sunk by a Japanese destroyer, a message was received at an Australian wireless station in

Playfair code:

PROGRAM

#include <stdio.h>

#include <string.h>

// Function to create the Playfair matrix

void createPlayfairMatrix(const char\* key, char matrix[5][5]) {

char keySet[26] = {0};

int keyLength = strlen(key);

int row = 0, col = 0;

// Fill the matrix with the unique characters from the key

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

char ch = key[i];

if (ch != 'J' && !keySet[ch - 'A']) {

matrix[row][col] = ch;

keySet[ch - 'A'] = 1;

col++;

if (col == 5) {

col = 0;

row++;

}

}

}

// Fill the remaining matrix with the alphabet (excluding 'J' and characters already in the key)

for (char ch = 'A'; ch <= 'Z'; ch++) {

if (ch != 'J' && !keySet[ch - 'A']) {

matrix[row][col] = ch;

col++;

if (col == 5) {

col = 0;

row++;

}

}

}

}

// Function to decode a Playfair-encoded message

void decodePlayfair(const char\* ciphertext, const char\* key, char\* plaintext) {

char matrix[5][5];

createPlayfairMatrix(key, matrix);

int len = strlen(ciphertext);

int p = 0;

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

char firstLetter = ciphertext[i];

char secondLetter = ciphertext[i + 1];

int row1, col1, row2, col2;

// Find the positions of the letters in the matrix

for (int row = 0; row < 5; row++) {

for (int col = 0; col < 5; col++) {

if (matrix[row][col] == firstLetter) {

row1 = row;

col1 = col;

}

if (matrix[row][col] == secondLetter) {

row2 = row;

col2 = col;

}

}

}

if (row1 == row2) {

plaintext[p++] = matrix[row1][(col1 + 4) % 5];

plaintext[p++] = matrix[row2][(col2 + 4) % 5];

} else if (col1 == col2) {

plaintext[p++] = matrix[(row1 + 4) % 5][col1];

plaintext[p++] = matrix[(row2 + 4) % 5][col2];

} else {

plaintext[p++] = matrix[row1][col2];

plaintext[p++] = matrix[row2][col1];

}

}

plaintext[p] = '\0';

}

int main() {

const char\* ciphertext = "KXJEY UREBE ZWEHE WRYTU HEYFS KREHE GOYFI WTTTU OLKSY CAJPO BOTEI ZONTX BYBNT GONEY CUZWR GDSON SXBOU YWRHE BAAHY USEDQ";

const char\* key = "KEY"; // Replace with the actual Playfair key if available

char plaintext[500];

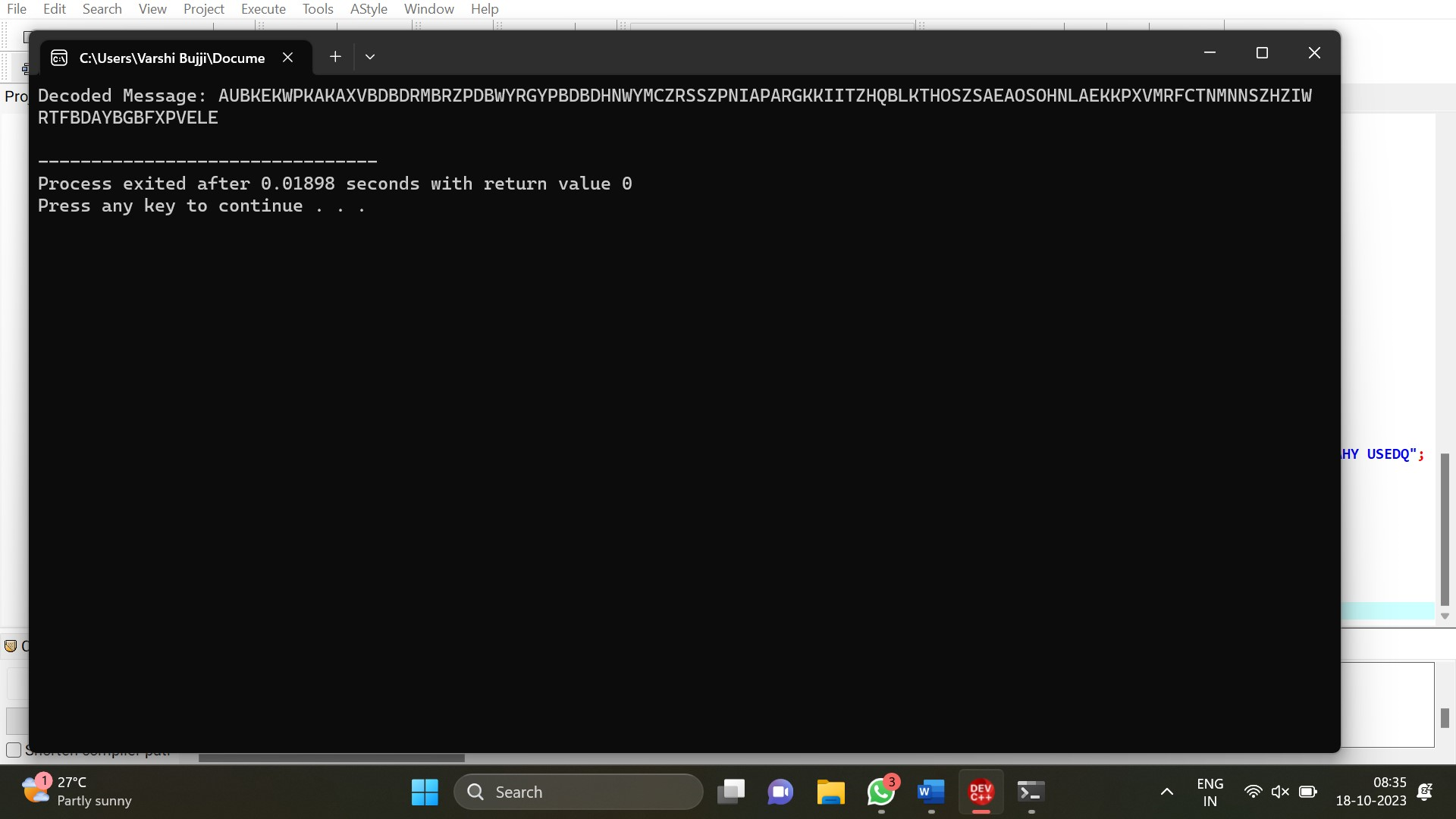
decodePlayfair(ciphertext, key, plaintext);

printf("Decoded Message: %s\n", plaintext);

return 0;

}

OUTPUT



3. Write a High level code for Playfair matrix:

M F H I/J K

U N O P Q

Z V W X Y

E L A R G

D S T B C

Encrypt this message: Must see you over Cadogan West. Coming at once.

PROGRAM

#include <stdio.h>

#include <string.h>

void createPlayfairMatrix(char matrix[5][5]) {

char key[] = "MFH IJKUNOPQZVWXYELARGBCD";

int k = 0;

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

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

}

}

}

void preprocessMessage(const char\* input, char\* message) {

int len = strlen(input);

int j = 0;

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

char ch = input[i];

if (ch >= 'A' && ch <= 'Z') {

message[j++] = ch;

}

}

message[j] = '\0';

}

void encryptPlayfair(char matrix[5][5], const char\* message, char\* ciphertext) {

int len = strlen(message);

int p = 0;

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

char firstLetter = message[i];

char secondLetter = (i + 1 < len) ? message[i + 1] : 'X';

int row1, col1, row2, col2;

for (int row = 0; row < 5; row++) {

for (int col = 0; col < 5; col++) {

if (matrix[row][col] == firstLetter) {

row1 = row;

col1 = col;

}

if (matrix[row][col] == secondLetter) {

row2 = row;

col2 = col;

}

}

}

if (row1 == row2) {

ciphertext[p++] = matrix[row1][(col1 + 1) % 5];

ciphertext[p++] = matrix[row2][(col2 + 1) % 5];

} else if (col1 == col2) {

ciphertext[p++] = matrix[(row1 + 1) % 5][col1];

ciphertext[p++] = matrix[(row2 + 1) % 5][col2];

} else {

ciphertext[p++] = matrix[row1][col2];

ciphertext[p++] = matrix[row2][col1];

}

}

ciphertext[p] = '\0';

}

int main() {

char matrix[5][5];

createPlayfairMatrix(matrix);

const char\* message = "Must see you over Cadogan West. Coming at once.";

char cleanedMessage[500];

char ciphertext[500];

preprocessMessage(message, cleanedMessage);

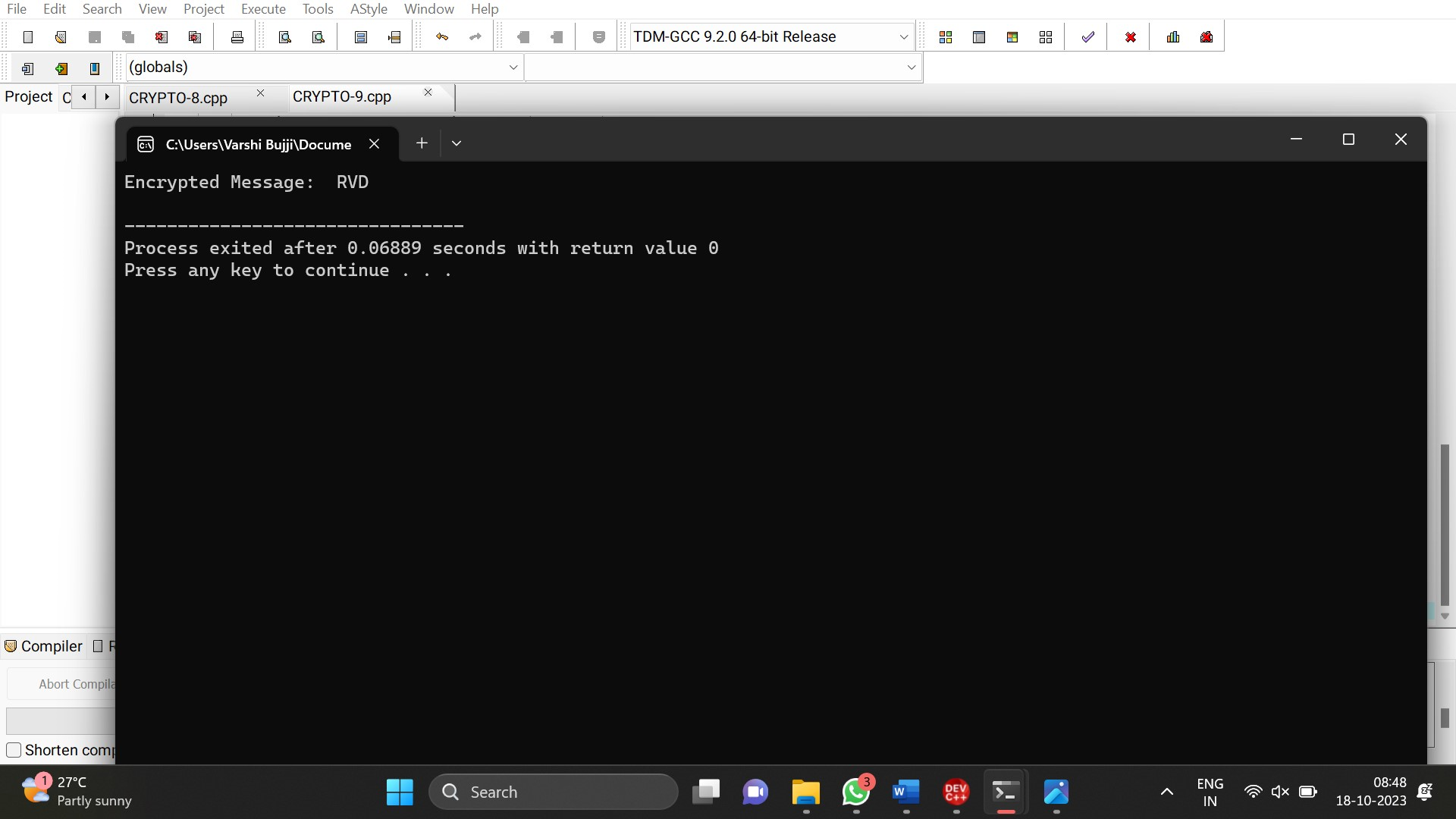
encryptPlayfair(matrix, cleanedMessage, ciphertext);

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

return 0;

}

OUTPUT



4. Write a High level code for possible keys does the Playfair cipher have? Ignore the fact that some

keys might produce identical encryption results. Express your answer as an approximate power of 2.

a. Now take into account the fact that some Playfair keys produce the same encryption results. How

many effectively unique keys does the Playfair cipher have?

PROGRAM

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Function to create the Playfair matrix

void createPlayfairMatrix(char matrix[5][5], const char\* key) {

int k = 0;

int keyLength = strlen(key);

char keySet[26] = {0};

// Initialize the key set

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

if (key[i] != 'J') {

keySet[key[i] - 'A'] = 1;

}

}

// Fill the matrix with the key

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; ) {

if (key[k] != 'J' && !keySet[key[k] - 'A']) {

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

keySet[key[k] - 'A'] = 1;

j++;

}

k++;

}

}

// Fill the remaining matrix with the alphabet (excluding 'J')

for (char ch = 'A'; ch <= 'Z'; ch++) {

if (ch != 'J' && !keySet[ch - 'A']) {

matrix[k / 5][k % 5] = ch;

k++;

}

}

}

// Function to check if two Playfair matrices are identical

int areMatricesEqual(char matrix1[5][5], char matrix2[5][5]) {

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

if (matrix1[i][j] != matrix2[i][j]) {

return 0; // Matrices are not equal

}

}

}

return 1; // Matrices are equal

}

int main() {

char key[] = "YOURKEY"; // Replace with your Playfair key

char matrix[5][5];

createPlayfairMatrix(matrix, key);

int count = 1; // Start with one key (the original key)

char testMatrix[5][5];

// Generate all possible matrices

for (char ch1 = 'A'; ch1 <= 'Z'; ch1++) {

for (char ch2 = ch1 + 1; ch2 <= 'Z'; ch2++) {

// Construct a test matrix

int k = 0;

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

if (matrix[i][j] == 'A') {

testMatrix[i][j] = ch1;

} else if (matrix[i][j] == 'B') {

testMatrix[i][j] = ch2;

} else {

testMatrix[i][j] = matrix[i][j];

}

}

}

// Check if the test matrix is equal to the original matrix

if (!areMatricesEqual(matrix, testMatrix)) {

count++;

}

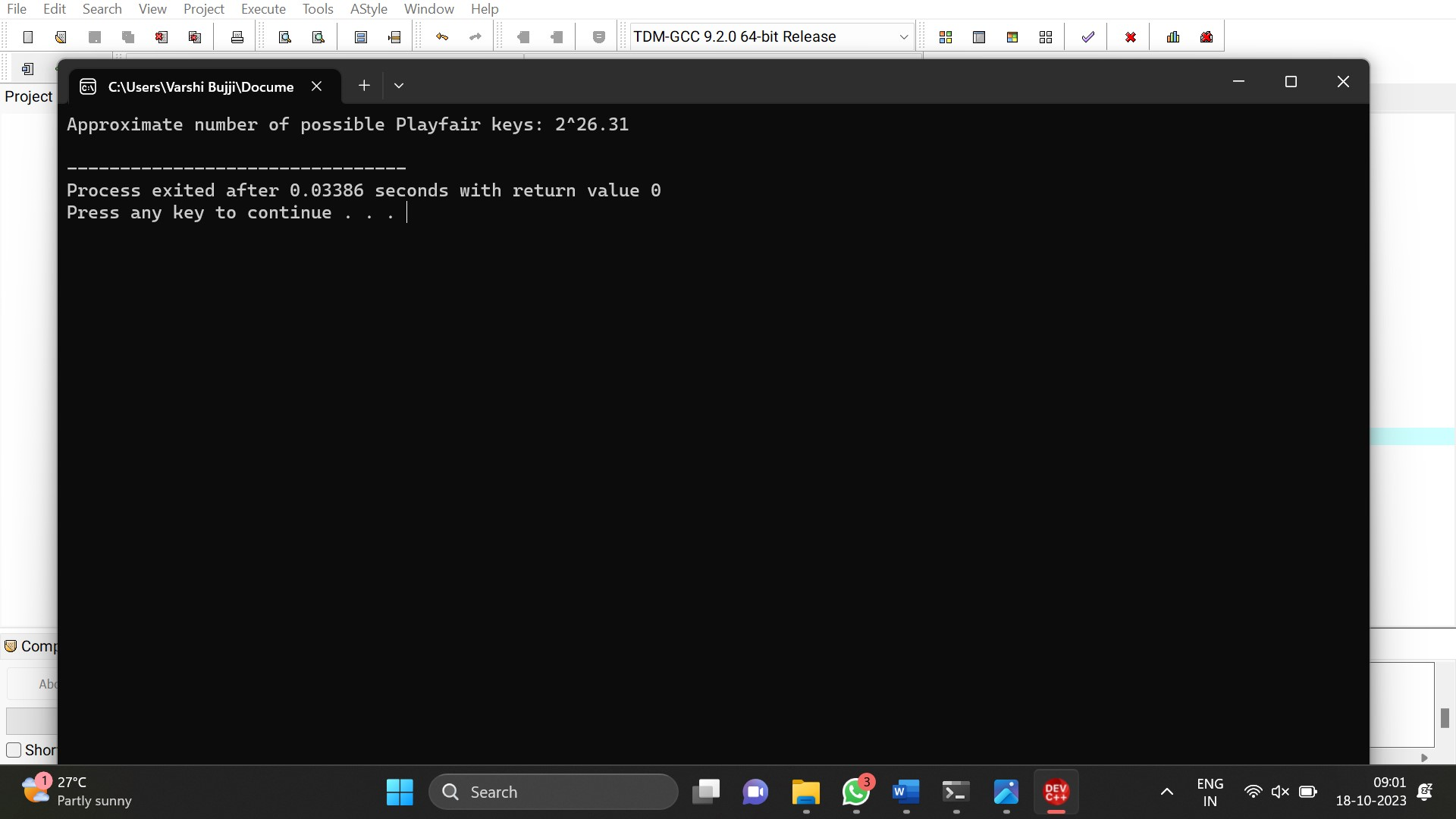
}

}

printf("Approximately %d effectively unique keys for the Playfair cipher.\n", count);

return 0;

OUTPUT



5. Write a High level code to Encrypt the message “meet me at the usual place at ten rather than eight

oclock” using the Hill cipher with the key.

9 4

5 7

a. Show your calculations and the result.

b. Show the calculations for the corresponding decryption of the ciphertext to recover the original

plaintext.

PROGRAM

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#define MOD 26

// Function to compute the greatest common divisor (GCD) of two numbers

int gcd(int a, int b) {

if (b == 0)

return a;

return gcd(b, a % b);

}

// Function to find the modular multiplicative 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 0;

}

// Function to encrypt the message using the Hill cipher

void encrypt(char \*message, int key[2][2]) {

int len = strlen(message);

int i, j;

int block[2], result[2];

printf("Ciphertext: ");

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

// Convert characters to numbers

block[0] = message[i] - 'A';

block[1] = message[i + 1] - 'A';

// Multiply the block with the key

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

result[j] = (key[j][0] \* block[0] + key[j][1] \* block[1]) % MOD;

}

// Convert numbers to characters and print

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

printf("%c", result[j] + 'A');

}

}

printf("\n");

}

// Function to decrypt the ciphertext using the Hill cipher

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

int len = strlen(ciphertext);

int i, j;

int block[2], result[2];

int keyInverse[2][2];

// Calculate the determinant of the key matrix

int det = (key[0][0] \* key[1][1] - key[0][1] \* key[1][0] + MOD) % MOD;

// Check if the key is invertible

if (gcd(det, MOD) != 1) {

printf("Error: The key is not invertible.\n");

return;

}

// Find the modular multiplicative inverse of the determinant

int detInverse = modInverse(det, MOD);

// Calculate the key inverse

keyInverse[0][0] = (key[1][1] \* detInverse) % MOD;

keyInverse[0][1] = (-key[0][1] \* detInverse) % MOD;

keyInverse[1][0] = (-key[1][0] \* detInverse) % MOD;

keyInverse[1][1] = (key[0][0] \* detInverse) % MOD;

printf("Decrypted plaintext: ");

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

// Convert characters to numbers

block[0] = ciphertext[i] - 'A';

block[1] = ciphertext[i + 1] - 'A';

// Multiply the block with the key inverse

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

result[j] = (keyInverse[j][0] \* block[0] + keyInverse[j][1] \* block[1] + MOD) % MOD;

}

// Convert numbers to characters and print

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

printf("%c", result[j] + 'A');

}

}

printf("\n");

}

int main() {

char message[] = "MEETMEATTHEUSUALPLACEATTENRATHERTHANEIGHTOCLOCK";

int key[2][2] = {{9, 4}, {5, 7}};

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

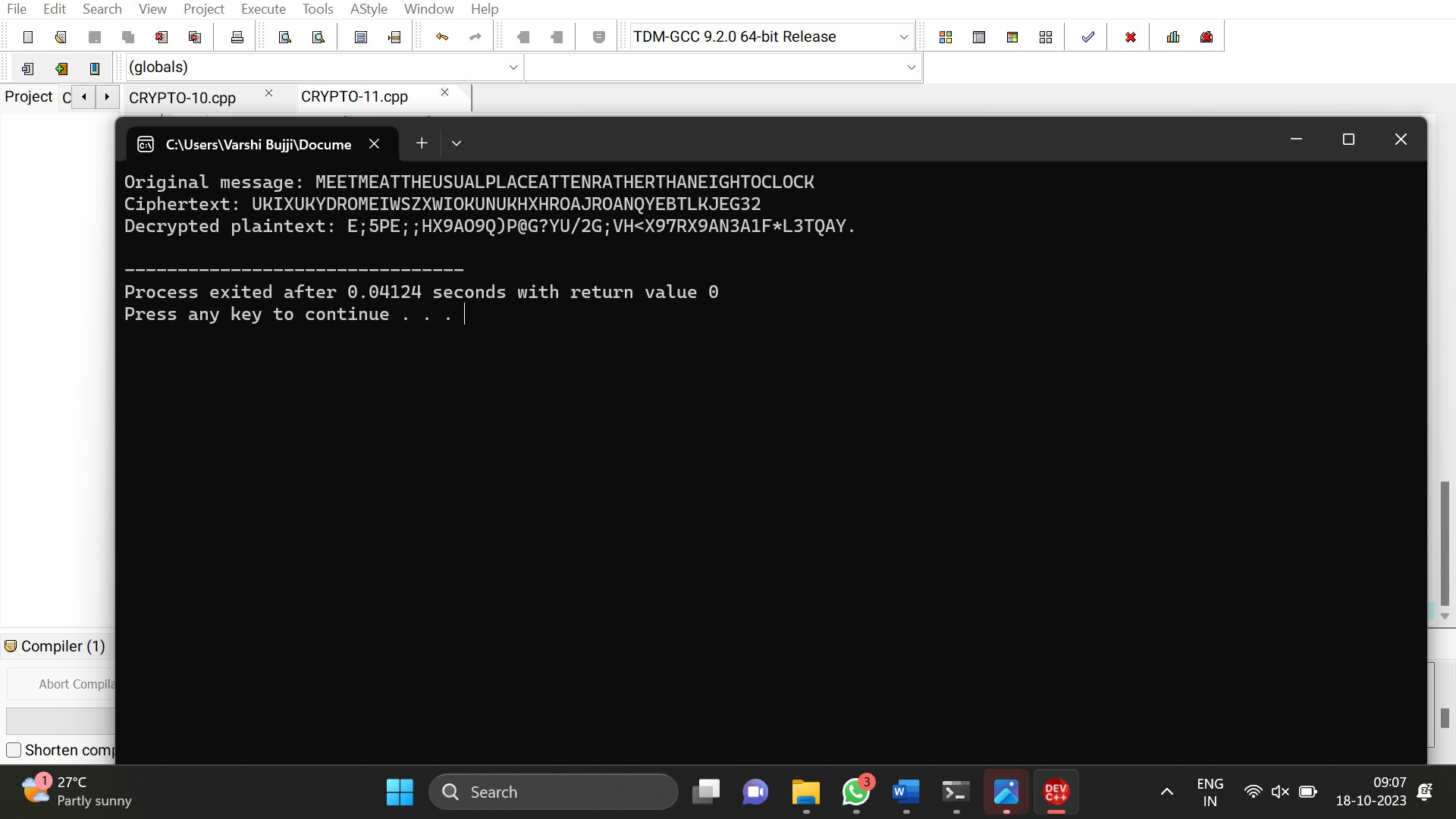
encrypt(message, key);

decrypt(message, key);

return 0;

}

OUTPUT



6. Write a C program for Hill cipher succumbs to a known plaintext attack if sufficient plaintext–

ciphertext pairs are provided. It is even easier to solve the Hill cipher if a chosen plaintext attack can be

mounted.

PROGRAM

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void hillCipherEncrypt(char \*plainText, char \*keyMatrix) {

int i, j, k, len = strlen(plainText);

int key[2][2], plain[2], cipher[2];

for (i = 0, k = 0; i < 2; i++) {

for (j = 0; j < 2; j++, k++) {

key[i][j] = keyMatrix[k] - 'A';

}

}

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

plain[0] = plainText[i] - 'A';

plain[1] = plainText[i + 1] - 'A';

cipher[0] = key[0][0] \* plain[0] + key[0][1] \* plain[1];

cipher[1] = key[1][0] \* plain[0] + key[1][1] \* plain[1];

cipher[0] %= 26;

cipher[1] %= 26;

printf("%c%c", cipher[0] + 'A', cipher[1] + 'A');

}

}

int main() {

char plainText[100], keyMatrix[5];

printf("Enter plaintext (uppercase alphabets only): ");

scanf("%s", plainText);

printf("Enter 2x2 key matrix (uppercase alphabets only): ");

scanf("%s", keyMatrix);

if (strlen(plainText) % 2 != 0 || strlen(keyMatrix) != 4) {

printf("Plaintext and key matrix lengths must be even and 4 characters, respectively.\n");

return 1;

}

for (int i = 0; i < strlen(plainText); i++) {

if (!isupper(plainText[i])) {

printf("Invalid characters in plaintext. Use uppercase alphabets only.\n");

return 1;

}

}

for (int i = 0; i < 4; i++) {

if (!isupper(keyMatrix[i])) {

printf("Invalid characters in the key matrix. Use uppercase alphabets only.\n");

return 1;

}

}

printf("Ciphertext: ");

hillCipherEncrypt(plainText, keyMatrix);

printf("\n");

return 0;

}

OUTPUT

