```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>


int determinant(int key[2][2]);
int modInverse(int a, int m);
void multiply(int key[2][2], int message[2], int ciphertext[2]);
void hillCipherEncrypt(char *plaintext, int key[2][2], char *ciphertext);


void inverseKey(int key[2][2], int inverseKeyMatrix[2][2]);
void hillCipherDecrypt(char *ciphertext, int inverseKeyMatrix[2][2], char
*plaintext);


void inverseKey(int key[2][2], int inverseKeyMatrix[2][2]) {
    int det = determinant(key);
    int det_mod_26 = det % 26;
    if (det_mod_26 < 0) {
        det_mod_26 += 26;
    }

    int invDet = modInverse(det_mod_26, 26);

    if (invDet != -1) {
        inverseKeyMatrix[0][0] = (key[1][1] * invDet) % 26;
        inverseKeyMatrix[0][1] = (-key[0][1] * invDet) % 26;
        inverseKeyMatrix[1][0] = (-key[1][0] * invDet) % 26;
        inverseKeyMatrix[1][1] = (key[0][0] * invDet) % 26;


        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                if (inverseKeyMatrix[i][j] < 0) {
                    inverseKeyMatrix[i][j] += 26;
                }
            }
        }
    } else {
        printf("La clÃ© n'est pas inversible modulo 26. Le dÃ©chiffrement
n'est pas possible.\n");
        inverseKeyMatrix[0][0] = inverseKeyMatrix[0][1] =
inverseKeyMatrix[1][0] = inverseKeyMatrix[1][1] = -1; // Indiquer une
clÃ© invalide
    }
}


void hillCipherDecrypt(char *ciphertext, int inverseKeyMatrix[2][2], char
*plaintext) {
    int len = strlen(ciphertext);
    int i, j = 0;

    if (len % 2 != 0) {
        printf("Le texte chiffrÃ© a une longueur impaire. Le
dÃ©chiffrement pourrait Ãªtre incorrect.\n");
```

```
        return;
    }

    for (i = 0; i < len; i += 2) {
        int message[2];
        message[0] = ciphertext[i] - 'a';
        message[1] = ciphertext[i + 1] - 'a';

        int decrypted[2];
        multiply(inverseKeyMatrix, message, decrypted);

        plaintext[j++] = decrypted[0] + 'a';
        plaintext[j++] = decrypted[1] + 'a';
    }
    plaintext[j] = '\0';
}

int main() {
    int choice;
    char text[100];
    int key[2][2];
    char result[100];
    int inverseKeyMatrix[2][2];

    do {
        printf("\nQue voulez-vous effectuer ?\n");
        printf("1. un Chiffrer un message\n");
        printf("2. un DÃ©chiffrement d'un message\n");
        printf("3. et Quitter\n");
        printf("Entrez votre choix : ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Entrez le texte clair (en minuscules) : ");
                scanf("%s", text);
                printf("Entrez la clÃ© de chiffrement 2x2 :\n");
                for (int i = 0; i < 2; i++) {
                    for (int j = 0; j < 2; j++) {
                        printf("key[%d][%d] = ", i, j);
                        scanf("%d", &key[i][j]);
                    }
                }
                hillCipherEncrypt(text, key, result);
                printf("Texte clair : %s\n", text);
                printf("Texte chiffrÃ© : %s\n", result);
                break;

            case 2:
                printf("Entrez le texte chiffrÃ© (en minuscules) : ");
                scanf("%s", text);
                printf("Entrez la clÃ© de dÃ©chiffrement 2x2 :\n");
                for (int i = 0; i < 2; i++) {
                    for (int j = 0; j < 2; j++) {
                        printf("key[%d][%d] = ", i, j);
                        scanf("%d", &key[i][j]);
                    }
                }
```

```
                inverseKey(key, inverseKeyMatrix);
                if (inverseKeyMatrix[0][0] != -1) {
                    hillCipherDecrypt(text, inverseKeyMatrix, result);
                    printf("Texte chiffrÃ© : %s\n", text);
                    printf("Texte dÃ©chiffrÃ© : %s\n", result);
                }
                break;

            case 3:
                printf("Au revoir !\n");
                break;

            default:
                printf("Choix invalide. Veuillez rÃ©essayer.\n");
        }
    } while (choice != 3);

    return 0;
}


int determinant(int key[2][2]) {
    return (key[0][0] * key[1][1]) - (key[0][1] * key[1][0]);
}

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;
}

void multiply(int key[2][2], int message[2], int ciphertext[2]) {
    ciphertext[0] = (key[0][0] * message[0] + key[0][1] * message[1]) %
26;
    ciphertext[1] = (key[1][0] * message[0] + key[1][1] * message[1]) %
26;
}

void hillCipherEncrypt(char *plaintext, int key[2][2], char *ciphertext)
{
    int len = strlen(plaintext);
    int i, j = 0;
    if (len % 2 != 0) {
        strcat(plaintext, "x");
        len++;
    }
    for (i = 0; i < len; i += 2) {
        int message[2];
        message[0] = plaintext[i] - 'a';
        message[1] = plaintext[i + 1] - 'a';
        int encrypted[2];
        multiply(key, message, encrypted);
        ciphertext[j++] = encrypted[0] + 'a';
        ciphertext[j++] = encrypted[1] + 'a';
```

```
    }
    ciphertext[j] = '\0';
}
```