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

#define MAX 10

// Fonction pour calculer le déterminant d'une matrice
int determinant(int matrix[MAX][MAX], int n) {
    int det = 0;
    int submatrix[MAX][MAX];
    if (n == 2) {
        return ((matrix[0][0] * matrix[1][1]) - (matrix[0][1] *
matrix[1][0]));
    }
    for (int x = 0; x < n; x++) {
        int subi = 0;
        for (int i = 1; i < n; i++) {
            int subj = 0;
            for (int j = 0; j < n; j++) {
                if (j == x)
                    continue;
                submatrix[subi][subj] = matrix[i][j];
                subj++;
            }
            subi++;
        }
        det = det + (x % 2 == 0 ? 1 : -1) * matrix[0][x] *
determinant(submatrix, n - 1);
    }
    return det;
}

// Fonction pour calculer l'inverse modulaire
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;
}

// Fonction pour calculer l'inverse d'une matrice
void inverseMatrix(int key[MAX][MAX], int n, int inverse[MAX][MAX]) {
    int det = determinant(key, n);
    det = det % 26;
    if (det < 0)
        det += 26;

    int detInv = modInverse(det, 26);
    if (detInv == -1) {
        printf("Erreur : La matrice clé n'est pas inversible dans
Z26.\n");
        exit(1);
    }

    int temp[MAX][MAX];
    if (n == 2) {
```

```
        temp[0][0] = key[1][1];
        temp[0][1] = -key[0][1];
        temp[1][0] = -key[1][0];
        temp[1][1] = key[0][0];
    } else {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                int submatrix[MAX][MAX];
                int subi = 0;
                for (int k = 0; k < n; k++) {
                    if (k == i)
                        continue;
                    int subj = 0;
                    for (int l = 0; l < n; l++) {
                        if (l == j)
                            continue;
                        submatrix[subi][subj] = key[k][l];
                        subj++;
                    }
                    subi++;
                }
                temp[j][i] = ((i + j) % 2 == 0 ? 1 : -1) *
determinant(submatrix, n - 1);
            }
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            inverse[i][j] = (temp[i][j] * detInv) % 26;
            if (inverse[i][j] < 0)
                inverse[i][j] += 26;
        }
    }
}

// Fonction pour multiplier une matrice par un vecteur
void multiplyMatrix(int matrix[MAX][MAX], int vector[MAX], int
result[MAX], int n) {
    for (int i = 0; i < n; i++) {
        result[i] = 0;
        for (int j = 0; j < n; j++) {
            result[i] += matrix[i][j] * vector[j];
        }
        result[i] %= 26;
    }
}

// Fonction pour chiffrer un texte
void hillEncrypt(char *plaintext, int key[MAX][MAX], int n) {
    int len = strlen(plaintext);
    if (len % n != 0) {
        printf("Le texte doit être un multiple de %d. Ajout de
remplissage 'X'.\n", n);
        while (len % n != 0) {
            plaintext[len] = 'X';
            len++;
        }
```

```
        plaintext[len] = '\0';
    }

    for (int i = 0; i < len; i += n) {
        int vector[MAX], result[MAX];
        for (int j = 0; j < n; j++) {
            vector[j] = plaintext[i + j] - 'A';
        }
        multiplyMatrix(key, vector, result, n);
        for (int j = 0; j < n; j++) {
            plaintext[i + j] = result[j] + 'A';
        }
    }
}

// Fonction pour déchiffrer un texte
void hillDecrypt(char *ciphertext, int key[MAX][MAX], int n) {
    int inverse[MAX][MAX];
    inverseMatrix(key, n, inverse);

    int len = strlen(ciphertext);
    for (int i = 0; i < len; i += n) {
        int vector[MAX], result[MAX];
        for (int j = 0; j < n; j++) {
            vector[j] = ciphertext[i + j] - 'A';
        }
        multiplyMatrix(inverse, vector, result, n);
        for (int j = 0; j < n; j++) {
            ciphertext[i + j] = result[j] + 'A';
        }
    }
}

// Fonction principale
int main() {
    char text[100];
    int key[MAX][MAX];
    int n, choice;
    int continueProgram = 1;

    printf("Entrer la taille de la matrice clé : ");
    scanf("%d", &n);

    printf("Entrer la matrice clé (%dx%d) :\n", n, n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &key[i][j]);
        }
    }

    printf("Entrer le texte en majuscules (A-Z, sans espaces) : ");
    scanf("%s", text);

    while (continueProgram) {
        printf("\nMenu :\n");
        printf("1. Chiffrer le texte\n");
        printf("2. Déchiffrer le texte\n");
        printf("3. Quitter\n");
```

```
        printf("Votre choix : ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                hillEncrypt(text, key, n);
                printf("Texte chiffré : %s\n", text);
                break;
            case 2:
                hillDecrypt(text, key, n);
                printf("Texte déchiffré : %s\n", text);
                break;
            case 3:
                continueProgram = 0;
                printf("Programme terminé.\n");
                break;
            default:
                printf("Choix invalide. Réessayez.\n");
        }

        if (continueProgram) {
            printf("\nVoulez-vous effectuer une autre opération sur le
texte actuel ? (1: Oui, 0: Non) : ");
            scanf("%d", &continueProgram);
        }
    }

    return 0;
}
```