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

// Définir la taille maximale du message
#define MAX 100

// Définir l'alphabet Z37 (a-z, 0-9, espace)
char alphabet[] = "abcdefghijklmnopqrstuvwxyz0123456789 ";

// --- FONCTIONS UTILITAIRES ---

// PGCD
int pgcd(int a, int b) {
    while (b != 0) {
        int r = a % b;
        a = b;
        b = r;
    }
    return a;
}

// Vérifie si un entier est inversible modulo n
int est_inversible(int a, int n) {
    return pgcd(a, n) == 1;
}

// Algorithme d'Euclide étendu
void bezout(int a, int b, int *u, int *v) {
    if (b == 0) {
        *u = 1;
        *v = 0;
        return;
    }
    int u1, v1;
    bezout(b, a % b, &u1, &v1);
    *u = v1;
    *v = u1 - (a / b) * v1;
}

// Modulo inverse via Bézout
int mod_inverse(int a, int mod) {
    int u, v;
    bezout(a, mod, &u, &v);
    int inverse = (u % mod + mod) % mod;
    return inverse;
}

// Fonction pour obtenir l'indice d'un caractère dans Z37
int char_to_int(char c) {
    for (int i = 0; i < 37; i++) {
        if (alphabet[i] == c)
            return i;
    }
    return -1; // caractère non trouvé
}

// Fonction pour obtenir le caractère correspondant à un entier dans Z37
```

```
char int_to_char(int n) {
    if (n >= 0 && n < 37)
        return alphabet[n];
    return '?'; // caractère invalide
}

// --- MATRICES ---

// Fonction pour calculer le déterminant d'une matrice m x m
int determinant(int **mat, int m, int mod) {
    if (m == 2)
        return (mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0]) % mod;
    else if (m == 3)
        return (mat[0][0]*(mat[1][1]*mat[2][2]-mat[1][2]*mat[2][1])
              - mat[0][1]*(mat[1][0]*mat[2][2]-mat[1][2]*mat[2][0])
              + mat[0][2]*(mat[1][0]*mat[2][1]-mat[1][1]*mat[2][0])) %
mod;
    else
        return 0; // non supporté
}

// Fonction pour calculer la matrice inverse modulo
void inverse_matrix(int **mat, int **inv, int m, int mod) {
    int det = determinant(mat, m, mod);
    det = (det + mod) % mod;

    printf(" Déterminant de la matrice : %d\n", det);
    if (!est_inversible(det, mod)) {
        printf("[ERREUR] Le déterminant n'est pas inversible modulo %d.
Pas d'inverse.\n", mod);
        exit(EXIT_FAILURE);
    }

    int det_inv = mod_inverse(det, mod);
    printf(" Inverse du déterminant modulo %d : %d\n", mod, det_inv);

    if (m == 2) {
        inv[0][0] = (mat[1][1] * det_inv) % mod;
        inv[0][1] = (-mat[0][1] * det_inv + mod) % mod;
        inv[1][0] = (-mat[1][0] * det_inv + mod) % mod;
        inv[1][1] = (mat[0][0] * det_inv) % mod;
    } else if (m == 3) {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                int a = (i+1)%3, b = (i+2)%3;
                int c = (j+1)%3, d = (j+2)%3;
                int cofactor = (mat[a][c]*mat[b][d] -
mat[a][d]*mat[b][c]) % mod;
                inv[j][i] = (cofactor * det_inv + mod) % mod; //
transposée
            }
        }
    }
}

// --- CHIFFREMENT / DÉCHIFFREMENT ---

void chiffrer_hill(char *message, int **mat, int m, char *output) {
```

```
    int len = strlen(message);
    int index = 0;

    while (len % m != 0) {
        message[len++] = ' ';
    }

    printf(" Message a chiffrer : %s\n", message);

    for (int i = 0; i < len; i += m) {
        for (int row = 0; row < m; row++) {
            int val = 0;
            for (int col = 0; col < m; col++) {
                int char_val = char_to_int(message[i + col]);
                val += mat[row][col] * char_val;
                printf(" matrice[%d][%d]=%d * char[%d]='%c'(%d) => %d\n",
                       row, col, mat[row][col], i+col, message[i+col],
char_val, mat[row][col]*char_val);
            }
            output[index++] = int_to_char(val % 37);
        }
    }
    output[index] = '\0';
}

void dechiffrer_hill(char *message, int **mat, int m, char *output) {
    int **inv = malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++)
        inv[i] = malloc(m * sizeof(int));

    inverse_matrix(mat, inv, m, 37);

    int len = strlen(message);
    int index = 0;

    for (int i = 0; i < len; i += m) {
        for (int row = 0; row < m; row++) {
            int val = 0;
            for (int col = 0; col < m; col++) {
                int char_val = char_to_int(message[i + col]);
                val += inv[row][col] * char_val;
                printf(" inverse[%d][%d]=%d * char[%d]='%c'(%d) => %d\n",
                       row, col, inv[row][col], i+col, message[i+col],
char_val, inv[row][col]*char_val);
            }
            output[index++] = int_to_char((val % 37 + 37) % 37);
        }
    }
    output[index] = '\0';

    for (int i = 0; i < m; i++)
        free(inv[i]);
    free(inv);
}

// --- MAIN PROGRAMME ---

int main() {
```

```
    int m;
    printf("Entrez la taille de la matrice (2 ou 3) : ");
    scanf("%d", &m);

    int **matrix = malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++)
        matrix[i] = malloc(m * sizeof(int));

    printf("Entrez la matrice de chiffrement (ligne par ligne) :\n");
    for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &matrix[i][j]);

    int choix;
    char message[MAX];
    char resultat[MAX];

    do {
        printf("\n--- MENU ---\n");
        printf("1. Chiffrer\n");
        printf("2. Déchiffrer\n");
        printf("3. Quitter\n");
        printf("Votre choix : ");
        scanf("%d", &choix);
        getchar();

        switch (choix) {
            case 1:
                printf("Entrez le message à chiffrer  : ");
                fgets(message, MAX, stdin);
                message[strcspn(message, "\n")] = 0;
                printf("\n Début du chiffrement Hill...\n");
                chiffrer_hill(message, matrix, m, resultat);
                printf(" Message chiffré : %s\n", resultat);
                break;
            case 2:
                printf("Entrez le message à déchiffrer : ");
                fgets(message, MAX, stdin);
                message[strcspn(message, "\n")] = 0;
                printf("\n Début du déchiffrement Hill...\n");
                dechiffrer_hill(message, matrix, m, resultat);
                printf(" Message déchiffré : %s\n", resultat);
                break;
            case 3:
                printf("Au revoir !\n");
                break;
            default:
                printf("Choix invalide.\n");
        }
    } while (choix != 3);

    for (int i = 0; i < m; i++)
        free(matrix[i]);
    free(matrix);

    return 0;
}
```