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

#define MOD 37

// Convertit un caractère en entier selon notre alphabet étendu
int char_to_int(char c) {
    if (isalpha(c)) return toupper(c) - 'A';       // A-Z -> 0-25
    if (c == ' ') return 26;                       // espace -> 26
    if (isdigit(c)) return c - '0' + 27;           // 0-9 -> 27-36
    return -1;                                     // caractère non
supporté
}

// Inverse : convertit un entier en caractère
char int_to_char(int x) {
    if (x >= 0 && x <= 25) return 'A' + x;
    if (x == 26) return ' ';
    if (x >= 27 && x <= 36) return '0' + (x - 27);
    return '?'; // caractère inconnu
}

// Modulo positif
int mod(int a, int b) {
    return (a % b + b) % b;
}

// Inverse modulaire : retourne l'inverse de a mod m
int mod_inverse(int a, int m) {
    a = mod(a, m);
    for (int x = 1; x < m; x++) {
        if (mod(a * x, m) == 1)
            return x;
    }
    return -1; // pas d'inverse
}

// Déterminant d'une matrice 2x2 ou 3x3
int determinant(int** mat, int n) {
    if (n == 2) {
        return mod(mat[0][0]*mat[1][1] - mat[0][1]*mat[1][0], MOD);
    } else if (n == 3) {
        int det =
            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]);
        return mod(det, MOD);
    }
    return -1;
}

// Calcul de l'inverse d'une matrice 2x2 ou 3x3
void inverse_matrix(int** key, int** inv, int n) {
    int det = determinant(key, n);
    int inv_det = mod_inverse(det, MOD);
    if (inv_det == -1) {
```

```
        printf("La matrice n'est pas inversible modulo %d.\n", MOD);
        exit(1);
    }

    if (n == 2) {
        inv[0][0] = mod(key[1][1] * inv_det, MOD);
        inv[0][1] = mod(-key[0][1] * inv_det, MOD);
        inv[1][0] = mod(-key[1][0] * inv_det, MOD);
        inv[1][1] = mod(key[0][0] * inv_det, MOD);
    } else if (n == 3) {
        int cof[3][3];

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                int sub[2][2], si = 0;

                for (int r = 0; r < 3; r++) {
                    if (r == i) continue;
                    int sj = 0;
                    for (int c = 0; c < 3; c++) {
                        if (c == j) continue;
                        sub[si][sj++] = key[r][c];
                    }
                    si++;
                }

                int minor = sub[0][0]*sub[1][1] - sub[0][1]*sub[1][0];
                cof[i][j] = mod(((i + j) % 2 == 0 ? 1 : -1) * minor,
MOD);
            }
        }

        // Transposée des cofacteurs * inverse du déterminant
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                inv[i][j] = mod(cof[j][i] * inv_det, MOD);
    }
}

// Produit matrice * vecteur
void multiply(int** key, int* vector, int* result, int n) {
    for (int i = 0; i < n; i++) {
        result[i] = 0;
        for (int j = 0; j < n; j++) {
            result[i] += key[i][j] * vector[j];
        }
        result[i] = mod(result[i], MOD);
    }
}

// Fonction principale de chiffrement/déchiffrement
void hill_cipher(char* text, char* result, int** key, int n) {
    int len = strlen(text);
    while (len % n != 0) {
        text[len++] = ' ';
        text[len] = '\0';
    }
```

```
    for (int i = 0; i < len; i += n) {
        int vec[n], res[n];
        for (int j = 0; j < n; j++) {
            int val = char_to_int(text[i + j]);
            if (val == -1) {
                printf("Caractère invalide dans le texte.\n");
                exit(1);
            }
            vec[j] = val;
        }

        multiply(key, vec, res, n);
        for (int j = 0; j < n; j++)
            result[i + j] = int_to_char(res[j]);
    }
    result[len] = '\0';
}

// Alloue dynamiquement une matrice carrée
int** alloc_matrix(int n) {
    int** mat = (int**) malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        mat[i] = (int*) malloc(n * sizeof(int));
    }
    return mat;
}

// Libère une matrice
void free_matrix(int** mat, int n) {
    for (int i = 0; i < n; i++) {
        free(mat[i]);
    }
    free(mat);
}

// Demande à l'utilisateur de saisir une matrice
void saisir_matrice(int** mat, int n) {
    printf("Entrez les %d éléments de la matrice (%dx%d) :\n", n * n, n,
n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &mat[i][j]);
    getchar(); // consomme le \n
}

// Programme principal
int main() {
    while (1) {
        printf("\nCHIFFREMENT DE HILL\n");
        printf("1. Chiffrer un message\n");
        printf("2. Déchiffrer un message\n");
        printf("3. Quitter\n");
        printf("Choix : ");

        int choix, taille;
        scanf("%d", &choix);
        getchar(); // consomme le \n
```

```
        if (choix == 3) {
            printf("Bye !\n");
            break;
        }

        if (choix < 1 || choix > 3) {
            printf("Option invalide.\n");
            continue;
        }

        printf("Taille de la matrice (2 ou 3) : ");
        scanf("%d", &taille);
        getchar();

        if (taille != 2 && taille != 3) {
            printf("Seules les tailles 2x2 et 3x3 sont acceptées pour le
moment.\n");
            continue;
        }

        int** cle = alloc_matrix(taille);
        saisir_matrice(cle, taille);

        char message[100], resultat[100];
        printf("Texte (A-Z, espace, 0-9) : ");
        fgets(message, sizeof(message), stdin);
        message[strcspn(message, "\n")] = '\0';

        if (choix == 1) {
            hill_cipher(message, resultat, cle, taille);
            printf("Message chiffré : %s\n", resultat);
        } else {
            int** inverse = alloc_matrix(taille);
            inverse_matrix(cle, inverse, taille);
            hill_cipher(message, resultat, inverse, taille);
            printf("Message déchiffré : %s\n", resultat);
            free_matrix(inverse, taille);
        }

        free_matrix(cle, taille);

        char reponse[10];
        printf("Souhaitez-vous recommencer ? (o/n) : ");
        fgets(reponse, sizeof(reponse), stdin);
        if (reponse[0] != 'o' && reponse[0] != 'O') {
            printf("Fin du programme.\n");
            break;
        }
    }

    return 0;
}
```