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

#define MAX 10
#define MOD 37

// Alphabet utilisé
const char alphabet[MOD + 1] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ";

// Trouver l'indice d'un caractère dans l'alphabet
int indiceCaractere(char c) {
    for (int i = 0; i < MOD; i++) {
        if (alphabet[i] == c) return i;
    }
    return -1;
}

// Convertir un indice en caractère
char caractereDepuisIndice(int index) {
    return alphabet[index % MOD];
}

// Calcul du pgcd et de l'inverse modulaire
int pgcdEtInverse(int a, int b, int *x, int *y) {
    if (a == 0) {
        *x = 0;
        *y = 1;
        return b;
    }
    int x1, y1;
    int d = pgcdEtInverse(b % a, a, &x1, &y1);
    *x = y1 - (b / a) * x1;
    *y = x1;
    return d;
}

int inverseModulaire(int a, int mod) {
    a = ((a % mod) + mod) % mod;
    int x, y;
    int d = pgcdEtInverse(a, mod, &x, &y);
    if (d != 1) return -1;
    return (x % mod + mod) % mod;
}

// Calcul du déterminant modulo MOD
int determinant(int matrice[MAX][MAX], int n) {
    if (n == 1) return matrice[0][0] % MOD;
    if (n == 2) return (matrice[0][0]*matrice[1][1] -
matrice[0][1]*matrice[1][0] + MOD) % MOD;

    int det = 0, mineur[MAX][MAX];
    for (int p = 0; p < n; p++) {
        int ligne = 0;
        for (int i = 1; i < n; i++) {
            int colonne = 0;
            for (int j = 0; j < n; j++) {
```

```
                if (j == p) continue;
                mineur[ligne][colonne++] = matrice[i][j];
            }
            ligne++;
        }
        int signe = (p % 2 == 0) ? 1 : -1;
        int sousDet = determinant(mineur, n - 1);
        det = (det + signe * matrice[0][p] * sousDet) % MOD;
        if (det < 0) det += MOD;
    }
    return det;
}

// Transposition de matrice
void transposer(int matrice[MAX][MAX], int n) {
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++) {
            int tmp = matrice[i][j];
            matrice[i][j] = matrice[j][i];
            matrice[j][i] = tmp;
        }
}

// Calcul des cofacteurs modulo MOD
void cofacteurs(int matrice[MAX][MAX], int temp[MAX][MAX], int n) {
    int signe, mineur[MAX][MAX];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int subi = 0;
            for (int x = 0; x < n; x++) {
                if (x == i) continue;
                int subj = 0;
                for (int y = 0; y < n; y++) {
                    if (y == j) continue;
                    mineur[subi][subj++] = matrice[x][y];
                }
                subi++;
            }
            signe = ((i + j) % 2 == 0) ? 1 : -1;
            temp[i][j] = (signe * determinant(mineur, n - 1)) % MOD;
            if (temp[i][j] < 0) temp[i][j] += MOD;
        }
    }
}

// Inversion d'une matrice modulo MOD
int inverserMatrice(int matrice[MAX][MAX], int inverse[MAX][MAX], int n)
{
    int det = determinant(matrice, n);
    int detInverse = inverseModulaire(det, MOD);
    if (detInverse == -1) return 0;

    int cof[MAX][MAX];
    cofacteurs(matrice, cof, n);
    transposer(cof, n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            inverse[i][j] = (cof[i][j] * detInverse) % MOD;
```

```
            if (inverse[i][j] < 0) inverse[i][j] += MOD;
        }
    return 1;
}

// Nettoyage et conversion du message en indices
void preparerMessage(char *message, int *indices, int *longueur) {
    int j = 0;
    for (int i = 0; message[i]; i++) {
        char c = toupper(message[i]);
        int idx = indiceCaractere(c);
        if (idx != -1) {
            indices[j++] = idx;
        }
    }
    *longueur = j;
}

// Chiffrement Hill
void chiffrerMessage(int matrice[MAX][MAX], int n, char *message) {
    int indices[100], longueur;
    preparerMessage(message, indices, &longueur);

    while (longueur % n != 0) indices[longueur++] = indiceCaractere(' ');

    printf("Message chiffré : ");
      for (int i = 0; i < longueur; i += n) {
        for (int ligne = 0; ligne < n; ligne++) {
            int valeur = 0;
            for (int col = 0; col < n; col++) {
                valeur += matrice[ligne][col] * indices[i + col];
            }
            valeur %= MOD;
            printf("%c", caractereDepuisIndice(valeur));
        }
    }
    printf("\n");
}

// Déchiffrement Hill
void dechiffrerMessage(int matrice[MAX][MAX], int n, char *message) {
    int inverse[MAX][MAX];
    if (!inverserMatrice(matrice, inverse, n)) {
        printf("La matrice n'est pas inversible modulo %d.\n", MOD);
        return;
    }

    int indices[100], longueur;
    preparerMessage(message, indices, &longueur);
    if (longueur % n != 0) {
        printf("Le message doit avoir une longueur multiple de %d
caractères.\n", n);
        return;
    }

    printf("Message déchiffré : ");
    for (int i = 0; i < longueur; i += n) {
        for (int ligne = 0; ligne < n; ligne++) {
```

```
            int valeur = 0;
            for (int col = 0; col < n; col++) {
                valeur += inverse[ligne][col] * indices[i + col];
            }
            valeur = (valeur % MOD + MOD) % MOD;
            printf("%c", caractereDepuisIndice(valeur));
        }
    }
    printf("\n");
}

// Menu principal
void afficherMenu() {

printf("============================================================\n");
    printf("***** Bienvenue dans le chiffrement de Hill modulo 37
*****\n");
    printf("Alphabet utilisé : A-Z + 0-9 + espace\n");
    printf("1. Chiffrer un message\n");
    printf("2. Déchiffrer un message\n");
    printf("3. Quitter\n");

printf("============================================================\n");
}

int main() {
    int matrice[MAX][MAX];
    int n, choix;
    char message[100];

    while (1) {
        afficherMenu();
        printf("Faites votre choix : ");
        if (scanf("%d", &choix) != 1) {
            printf("Entrée invalide.\n");
            while (getchar() != '\n');
            continue;
        }
        while (getchar() != '\n');

        if (choix == 3) {
            printf("Au revoir !\n");
            break;
        }

        printf("Entrez la taille de la matrice carrée (max %d) : ", MAX);
        if (scanf("%d", &n) != 1 || n < 1 || n > MAX) {
            printf("Taille invalide.\n");
            while (getchar() != '\n');
            continue;
        }
        while (getchar() != '\n');

        printf("Entrez les éléments de la matrice (%d x %d), ligne par
ligne :\n", n, n);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) {
                if (scanf("%d", &matrice[i][j]) != 1) {
```

```
                printf("Entrée invalide.\n");
                while (getchar() != '\n');
                continue;
            }
        }
        while (getchar() != '\n');

        printf("Entrez le message : ");
        if (fgets(message, sizeof(message), stdin) == NULL) {
            printf("Erreur lors de la lecture du message.\n");
            continue;
        }
        message[strcspn(message, "\n")] = '\0';

        if (choix == 1)
            chiffrerMessage(matrice, n, message);
        else if (choix == 2)
            dechiffrerMessage(matrice, n, message);
        else
            printf("Choix invalide.\n");
    }

    return 0;
}
```