```
#include <stdio.h>
#include <string.h>
// Function to calculate modulo
int mod(int a, int b) {
   return (a % b + b) % b;
}
// Function to find the determinant of a 2x2 matrix
int determinant(int matrix[2][2]) {
   return (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]);
}
// Function to find modular multiplicative inverse
int modInverse(int a, int m) {
   for (int x = 1; x < m; x++) {
      if ((a * x) % m == 1) {
         return x;
      }
   }
   return -1; // No modular inverse exists
}
// Function to find the inverse of a 2x2 matrix modulo 26
int inverseMatrix(int matrix[2][2], int inverse[2][2]) {
   int det = determinant(matrix);
   int detMod = mod(det, 26);
   int detInverse = modInverse(detMod, 26);
   if (detInverse == -1) {
      return 0; // Matrix is not invertible
   }
  // Compute the adjoint matrix
```

```
    inverse[0][0] = matrix[1][1];
    inverse[0][1] = -matrix[0][1];
    inverse[1][0] = -matrix[1][0];
    inverse[1][1] = matrix[0][0];
    // Multiply adjoint by determinant inverse and take modulo 26
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            inverse[i][j] = mod(inverse[i][j] * detInverse, 26);
        } }
    return 1; // Inverse exists
}
// Function to multiply matrices for encryption/decryption
void matrixMultiply(int matrix[2][2], int vector[2], int result[2]) {
    for (int i = 0; i < 2; i++) {
        result[i] = 0;
        for (int j = 0; j < 2; j++) {
            result[i] += matrix[i][j] * vector[j];
        }
        result[i] = mod(result[i], 26); // Mod 26 for alphabet (A-Z)
    }}
int main() {
    char plaintext[3], ciphertext[3];
    int keyMatrix[2][2], inverseKey[2][2];
    // Input plaintext
    printf("Enter a 2-letter plaintext (uppercase only): ");
    scanf("%2s", plaintext);
    if (strlen(plaintext) != 2) {
        printf("Plaintext must be exactly 2 letters!\n");
```

```
    return 1;
}
// Input the 2x2 key matrix
printf("Enter the 2x2 key matrix (4 numbers):\n");
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        scanf("%d", &keyMatrix[i][j]);
    } }
// Check if the matrix is invertible
if (!inverseMatrix(keyMatrix, inverseKey)) {
    printf("The key matrix is not invertible. Decryption is not possible.\n");
    return 1;
}
// Convert plaintext to numerical vector
int plaintextVector[2], ciphertextVector[2];
plaintextVector[0] = plaintext[0] - 'A';
plaintextVector[1] = plaintext[1] - 'A';
// Encrypt plaintext
matrixMultiply(keyMatrix, plaintextVector, ciphertextVector);
// Convert ciphertext vector to characters
ciphertext[0] = ciphertextVector[0] + 'A';
ciphertext[1] = ciphertextVector[1] + 'A';
ciphertext[2] = '\0';
printf("Encrypted ciphertext: %s\n", ciphertext);
// Decrypt ciphertext
int decryptedVector[2];
matrixMultiply(inverseKey, ciphertextVector, decryptedVector);
```

```
// Convert decrypted vector to plaintext
char decryptedText[3];
decryptedText[0] = decryptedVector[0] + 'A';
decryptedText[1] = decryptedVector[1] + 'A';
decryptedText[2] = '\0';
printf("Decrypted plaintext: %s\n", decryptedText);
return 0;
                    }
```