main.c

Run

Output

Clear

```c
#include <stdio.h>

#define MOD 26

// Helper to find modular inverse of a number
    modulo 26
int modInverse(int a) {
    a = a % MOD;
    for (int x = 1; x < MOD; x++) {
        if ((a * x) % MOD == 1)
            return x;
    }
    return -1; // No inverse
}

// Find inverse of 2x2 matrix mod 26
int findMatrixInverse(int P[2][2], int
    InvP[2][2]) {
```

```
Recovered Key Matrix (K):
4 5
25 19


=== Code Execution Successful ===
```

main.c

Run

Output

Clear

```
13  }
14
15  // Find inverse of 2x2 matrix mod 26
16  int findMatrixInverse(int P[2][2], int
        InvP[2][2]) {
17      int det = (P[0][0] * P[1][1] - P[0][1] *
            P[1][0]) % MOD;
18      if (det < 0) det += MOD;
19
20      int invDet = modInverse(det);
21      if (invDet == -1) {
22          printf("Matrix is not invertible!\n");
23          return 0;
24      }
25
26      // Inverse matrix formula
27      InvP[0][0] = ( P[1][1] * invDet) % MOD;
28      InvP[0][1] = (-P[0][1] * invDet) % MOD;
```

```
Recovered Key Matrix (K):
4 5
25 19


=== Code Execution Successful ===
```

main.c

Run

```
    };

    int InvP[2][2];
    if (!findMatrixInverse(P, InvP)) return 1;

    int K[2][2];
    multiplyMatrices(C, InvP, K);

    printf("Recovered Key Matrix (K):\n");
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++)
            printf("%d ", K[i][j]);
        printf("\n");
    }

    return 0;
}
```

Output

Clear

```
Recovered Key Matrix (K):
4 5
25 19


=== Code Execution Successful ===
```