# Assignment 2

IT851: Information and Systems Security Lab

**ANINDYA KUNDU**

IT, 8th Semester

ID 510817020

Repository:
github.com/meganindya/btech-assignments/information-and-systems-security/assg-2

Implement the following traditional symmetric ciphers:

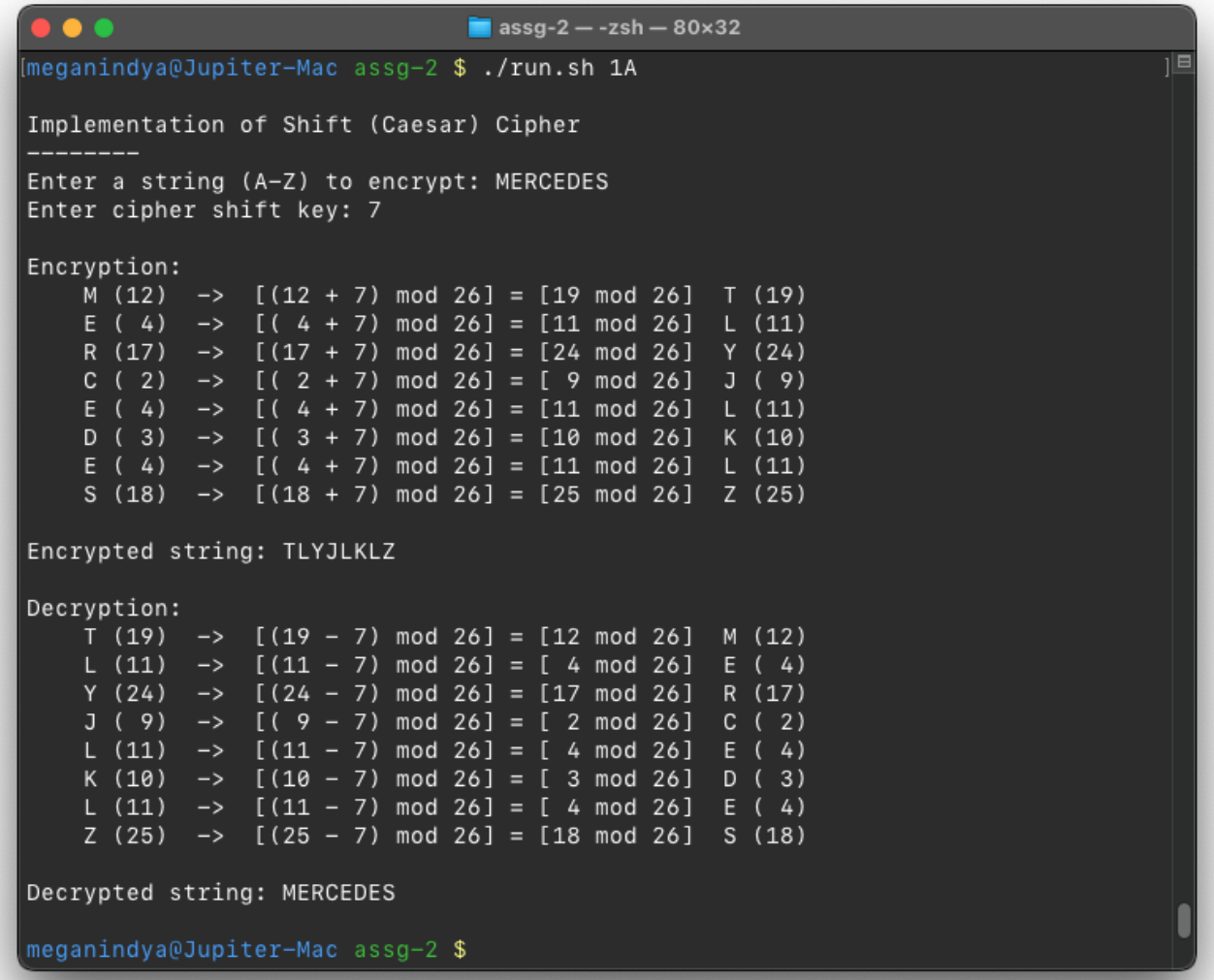
1. Shift Cipher

Source: `1-a-cipher-shift.c`

```c
#include <stdio.h>  // printf, scanf
#include <stdlib.h> // abs

#include "utils.h"
#define MOD 26

/*
 * Cipher shifts (in place) all characters of a string by a distance.
 *
 * s: the string
 * z: shift distance
 */
void encrypt(char *s, int z)
{
    for (int i = 0; s[i] != '\0'; i++)
    {
        char c = s[i];
        int c_n = c - 'A';
        int add_n = c_n + z;
        int n_enc = mod_26(add_n);
        char c_enc = n_enc + 'A';
        printf(
            "    %c (%2d)  ->  [(%2d + %d) mod %d] = [%2d mod %d]  %c (%2d)\n",
            c,
            c_n,
            c_n,
            z,
            MOD,
            add_n,
            MOD,
            c_enc,
            n_enc);
        s[i] = c_enc;
    }
}

/*
 * Cipher unshifts (in place) all characters of a string by a distance.
 *
 * s: the string
 * z: shift distance
 */
void decrypt(char *s, int z)
{
    for (int i = 0; s[i] != '\0'; i++)
    {
        char c = s[i];
```

```
        int c_n = c - 'A';
        int sub_n = c_n - z;
        int n_dec = mod_26(sub_n);
        char c_dec = n_dec + 'A';
        printf(
            "    %c (%2d)  ->  [(%2d - %d) mod %d] = [%2d mod %d]  %c (%2d)\n",
            c,
            c_n,
            c_n,
            z,
            MOD,
            sub_n,
            MOD,
            c_dec,
            n_dec);
        s[i] = c_dec;
    }
}

// ---------------------------------------------------------------------------

int main(int argc, char *argv[])
{
    char s[256];
    int z;
    printf("\nImplementation of Shift (Caesar) Cipher\n--------\n");
    int repeat;
    do
    {
        printf("Enter a string (A-Z) to encrypt: ");
        scanf("%s", s);
        repeat = 0;
        for (int i = 0; s[i] != '\0'; i++)
        {
            if (s[i] < 'A' || s[i] > 'Z')
            {
                printf("  Invalid string, retry\n");
                repeat = 1;
                break;
            }
        }
    } while (repeat);
    printf("Enter cipher shift key: ");
    scanf("%d", &z);
    printf("\nEncryption:\n");
    encrypt(s, z);
    printf("\nEncrypted string: %s\n", s);
    printf("\nDecryption:\n");
    decrypt(s, z);
    printf("\nDecrypted string: %s\n", s);
    printf("\n");
}
```

Sample run




2. Multiplicative cipher

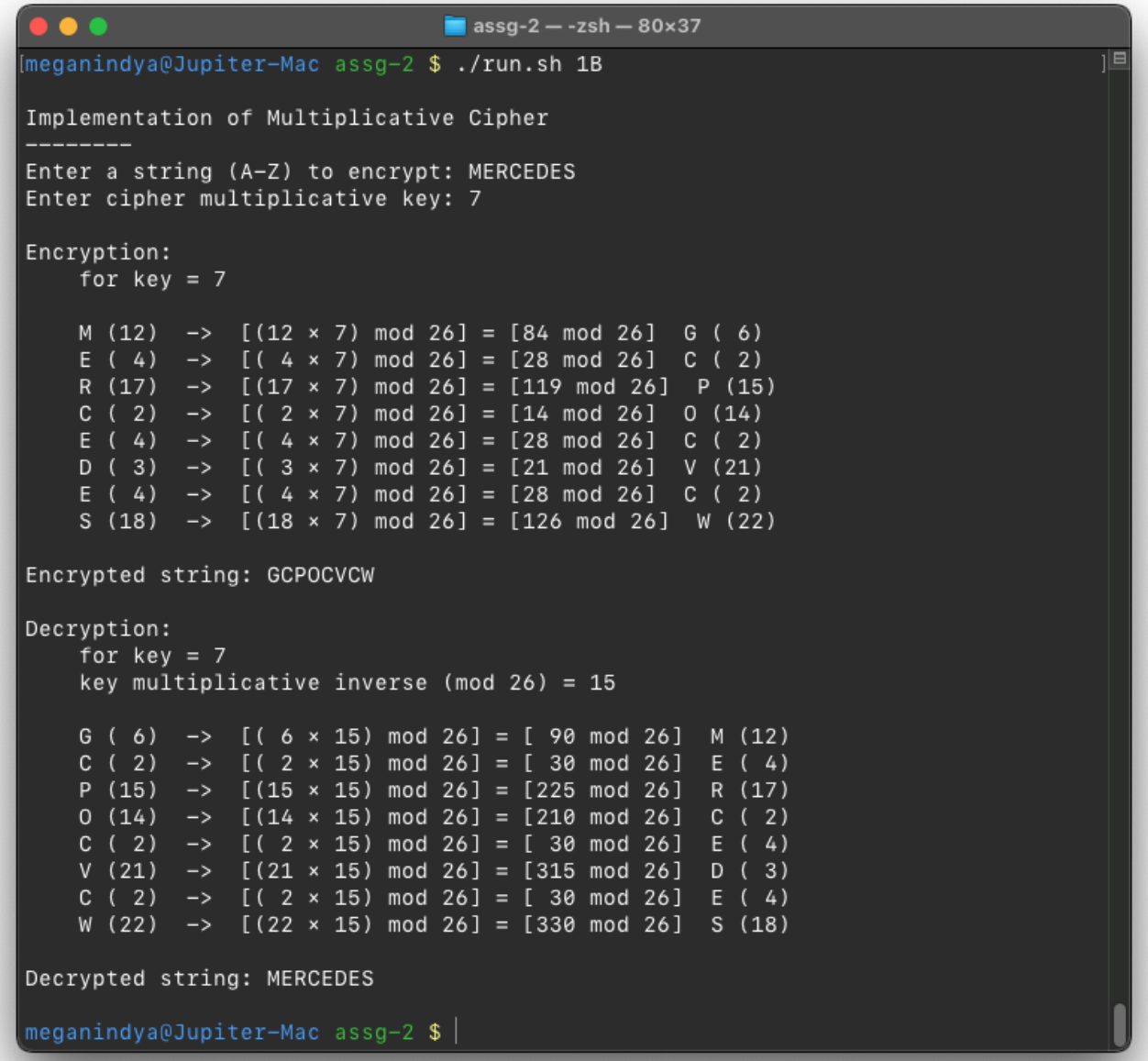
Source: `1-b-cipher-multiplicative.c`

```c
#include <stdio.h>  // printf, scanf
#include <stdlib.h> // abs

#include "utils.h"
#define MOD 26

/*
 * Cipher multiplicative encrypts (in place) all characters of a string w.r.t a key.
 *
 * s: the string
 * key: cipher key
 */
void encrypt(char *s, int key)
```

```
{
    printf("    for key = %d\n\n", key);
    for (int i = 0; s[i] != '\0'; i++)
    {
        char enc_char = 'A' + mod_26((s[i] - 'A') * key);
        printf(
            "    %c (%2d)  ->  [(%2d × %d) mod %d] = [%2d mod %d]  %c (%2d)\n",
            s[i],
            s[i] - 'A',
            s[i] - 'A',
            key,
            MOD,
            (s[i] - 'A') * key,
            MOD,
            enc_char,
            enc_char - 'A');
        s[i] = enc_char;
    }
}

/*
 * Cipher multiplicative decrypts (in place) all characters of a string w.r.t a key.
 *
 * s: the string
 * key: cipher key
 */
void decrypt(char *s, int key)
{
    int key_inv = mod_26_mul_inv(key);
    printf("    for key = %d\n    key multiplicative inverse (mod %d) = %d\n\n", key, MOD,
 key_inv);
    for (int i = 0; s[i] != '\0'; i++)
    {
        char dec_char = 'A' + mod_26((s[i] - 'A') * key_inv);
        printf(
            "    %c (%2d)  ->  [(%2d × %d) mod %d] = [%3d mod %d]  %c (%2d)\n",
            s[i],
            s[i] - 'A',
            s[i] - 'A',
            key_inv,
            MOD,
            (s[i] - 'A') * key_inv,
            MOD,
            dec_char,
            dec_char - 'A');
        s[i] = dec_char;
    }
}

// ------------------------------------------------------------------------

int main(int argc, char *argv[])
```

```
{
    char s[256];
    int key;
    printf("\nImplementation of Multiplicative Cipher\n--------\n");
    int repeat;
    do
    {
        printf("Enter a string (A-Z) to encrypt: ");
        scanf("%s", s);
        repeat = 0;
        for (int i = 0; s[i] != '\0'; i++)
        {
            if (s[i] < 'A' || s[i] > 'Z')
            {
                printf("  Invalid string, retry\n");
                repeat = 1;
                break;
            }
        }
    } while (repeat);
    do
    {
        printf("Enter cipher multiplicative key: ");
        scanf("%d", &key);
        repeat = 0;
        if (mod_26_mul_inv(key) == -1)
        {
            printf("  Invalid key, retry\n");
            repeat = 1;
        }
    } while (repeat);
    printf("\nEncryption:\n");
    encrypt(s, key);
    printf("\nEncrypted string: %s\n", s);
    printf("\nDecryption:\n");
    decrypt(s, key);
    printf("\nDecrypted string: %s\n", s);
    printf("\n");
}
```

Sample run


assg-2 — -zsh — 80×37
meganindya@Jupiter-Mac assg-2 $ ./run.sh 1B
Implementation of Multiplicative Cipher
--------
Enter a string (A-Z) to encrypt: MERCEDES
Enter cipher multiplicative key: 7
Encryption:
for key = 7
M (12) -> [(12 × 7) mod 26] = [84 mod 26] G ( 6)
E ( 4) -> [( 4 × 7) mod 26] = [28 mod 26] C ( 2)
R (17) -> [(17 × 7) mod 26] = [119 mod 26] P (15)
C ( 2) -> [( 2 × 7) mod 26] = [14 mod 26] O (14)
E ( 4) -> [( 4 × 7) mod 26] = [28 mod 26] C ( 2)
D ( 3) -> [( 3 × 7) mod 26] = [21 mod 26] V (21)
E ( 4) -> [( 4 × 7) mod 26] = [28 mod 26] C ( 2)
S (18) -> [(18 × 7) mod 26] = [126 mod 26] W (22)
Encrypted string: GCPOCVCW
Decryption:
for key = 7
key multiplicative inverse (mod 26) = 15
G ( 6) -> [( 6 × 15) mod 26] = [ 90 mod 26] M (12)
C ( 2) -> [( 2 × 15) mod 26] = [ 30 mod 26] E ( 4)
P (15) -> [(15 × 15) mod 26] = [225 mod 26] R (17)
O (14) -> [(14 × 15) mod 26] = [210 mod 26] C ( 2)
C ( 2) -> [( 2 × 15) mod 26] = [ 30 mod 26] E ( 4)
V (21) -> [(21 × 15) mod 26] = [315 mod 26] D ( 3)
C ( 2) -> [( 2 × 15) mod 26] = [ 30 mod 26] E ( 4)
W (22) -> [(22 × 15) mod 26] = [330 mod 26] S (18)
Decrypted string: MERCEDES
meganindya@Jupiter-Mac assg-2 $

3. Affine Cipher

Source: `1-c-cipher-affine.c`

```c
#include <stdio.h>  // printf, scanf
#include <stdlib.h> // abs

#include "utils.h"
#define MOD 26

/*
 * Cipher multiplicative encrypts (in place) all characters of a string w.r.t a key.
 *
 * s: the string
 * key: cipher key
 */
void encrypt(char *s, int key_mul, int key_add)
{
    printf("    for key (a, b) = (%d, %d)\n\n", key_mul, key_add);
    for (int i = 0; s[i] != '\0'; i++)
    {
        char enc_char = 'A' + mod_26((s[i] - 'A') * key_mul + key_add);
        printf(
            "    %c (%2d)  ->  [(%2d × %d + %d) mod %d] = [%2d mod %d]  %c (%2d)\n",
            s[i],
            s[i] - 'A',
            s[i] - 'A',
            key_mul,
            key_add,
            MOD,
            (s[i] - 'A') * key_mul + key_add,
            MOD,
            enc_char,
            enc_char - 'A');
        s[i] = enc_char;
    }
}

/*
 * Cipher multiplicative decrypts (in place) all characters of a string w.r.t a key.
 *
 * s: the string
 * key: cipher key
 */
void decrypt(char *s, int key_mul, int key_add)
{
    int key_add_inv = mod_26_add_inv(key_add);
    int key_mul_inv = mod_26_mul_inv(key_mul);
    printf("    for key (a, b) = (%d, %d)\n", key_mul, key_add);
    printf("    key (a = %d) multiplicative inverse (mod %d) = %d\n", key_mul, MOD, key_mu
l_inv);
    printf("    key (b = %d) additive inverse (mod %d) = %d\n\n", key_add, MOD, key_add_in
v);
```

```
    for (int i = 0; s[i] != '\0'; i++)
    {
        char dec_char = 'A' + mod_26(((s[i] - 'A') + key_add_inv) * key_mul_inv);
        printf(
            "    %c (%2d)  ->  [((%2d %d) × %d) mod %d] = [%3d mod %d]  %c (%2d)\n",
            s[i],
            s[i] - 'A',
            s[i] - 'A',
            key_add_inv,
            key_mul_inv,
            MOD,
            ((s[i] - 'A') + key_add_inv) * key_mul_inv,
            MOD,
            dec_char,
            dec_char - 'A');
        s[i] = dec_char;
    }
}

// ------------------------------------------------------------------------------

int main(int argc, char *argv[])
{
    char s[256];
    int key_add, key_mul;
    printf("\nImplementation of Affine Cipher\n--------\n");
    int repeat;
    do
    {
        printf("Enter a string (A-Z) to encrypt: ");
        scanf("%s", s);
        repeat = 0;
        for (int i = 0; s[i] != '\0'; i++)
        {
            if (s[i] < 'A' || s[i] > 'Z')
            {
                printf("  Invalid string, retry\n");
                repeat = 1;
                break;
            }
        }
    } while (repeat);
    printf("Enter cipher additive key: ");
    scanf("%d", &key_add);
    do
    {
        printf("Enter cipher multiplicative key: ");
        scanf("%d", &key_mul);
        repeat = 0;
        if (mod_26_mul_inv(key_mul) == -1)
        {
            printf("  Invalid key, retry\n");
```

```
            repeat = 1;
        }
    } while (repeat);
    printf("\nEncryption:\n");
    encrypt(s, key_mul, key_add);
    printf("\nEncrypted string: %s\n", s);
    printf("\nDecryption:\n");
    decrypt(s, key_mul, key_add);
    printf("\nDecrypted string: %s\n", s);
    printf("\n");
}
```

Sample run

```
assg-2 — -zsh — 80×39
meganindya@Jupiter-Mac assg-2 $ ./run.sh 1C

Implementation of Affine Cipher
--------
Enter a string (A-Z) to encrypt: MERCEDES
Enter cipher additive key: 2
Enter cipher multiplicative key: 7

Encryption:
    for key (a, b) = (7, 2)

    M (12)  ->  [(12 × 7 + 2) mod 26] = [86 mod 26]  I ( 8)
    E ( 4)  ->  [( 4 × 7 + 2) mod 26] = [30 mod 26]  E ( 4)
    R (17)  ->  [(17 × 7 + 2) mod 26] = [121 mod 26]  R (17)
    C ( 2)  ->  [( 2 × 7 + 2) mod 26] = [16 mod 26]  Q (16)
    E ( 4)  ->  [( 4 × 7 + 2) mod 26] = [30 mod 26]  E ( 4)
    D ( 3)  ->  [( 3 × 7 + 2) mod 26] = [23 mod 26]  X (23)
    E ( 4)  ->  [( 4 × 7 + 2) mod 26] = [30 mod 26]  E ( 4)
    S (18)  ->  [(18 × 7 + 2) mod 26] = [128 mod 26]  Y (24)

Encrypted string: IERQEXEY

Decryption:
    for key (a, b) = (7, 2)
    key (a = 7) multiplicative inverse (mod 26) = 15
    key (b = 2) additive inverse (mod 26) = -2

    I ( 8)  ->  [(( 8 -2) × 15) mod 26] = [ 90 mod 26]  M (12)
    E ( 4)  ->  [(( 4 -2) × 15) mod 26] = [ 30 mod 26]  E ( 4)
    R (17)  ->  [((17 -2) × 15) mod 26] = [225 mod 26]  R (17)
    Q (16)  ->  [((16 -2) × 15) mod 26] = [210 mod 26]  C ( 2)
    E ( 4)  ->  [(( 4 -2) × 15) mod 26] = [ 30 mod 26]  E ( 4)
    X (23)  ->  [((23 -2) × 15) mod 26] = [315 mod 26]  D ( 3)
    E ( 4)  ->  [(( 4 -2) × 15) mod 26] = [ 30 mod 26]  E ( 4)
    Y (24)  ->  [((24 -2) × 15) mod 26] = [330 mod 26]  S (18)

Decrypted string: MERCEDES

meganindya@Jupiter-Mac assg-2 $
```

4. Playfair Cipher

Source: `1-d-cipher-playfair.c`

```c
#include <stdio.h>  // printf, scanf
#include <stdlib.h> // abs
#include <string.h> // strlen

#include "utils.h"
#define MOD 26

/*
 * Cipher playfair encrypts (in place) all characters of a string w.r.t a key.
 *
 * s: the string
 * key: cipher key matrix
 */
void encrypt(char *s, int **key)
{
    int len = strlen(s);
    if ((len & 1) == 1)
    {
        printf("  Salting odd length string with 'Z' at end\n\n");
        s[len] = 'Z';
        s[len + 1] = '\0';
    }

    int map[26][2];
    map[9][0] = -1; // row of 'J'
    map[9][1] = -1; // col of 'J'
    for (int r = 0; r < 5; r++)
    {
        for (int c = 0; c < 5; c++)
        {
            map[key[r][c]][0] = r;
            map[key[r][c]][1] = c;
        }
    }

    printf("  Pairwise transformation:\n");
    for (int i = 0; s[i] != '\0'; i += 2)
    {
        int a_r = map[s[i] - 'A'][0];
        int a_c = map[s[i] - 'A'][1];
        int b_r = map[s[i + 1] - 'A'][0];
        int b_c = map[s[i + 1] - 'A'][1];

        int r_a, r_b;

        if (a_r == b_r)
        {
            r_a = key[a_r][mod(a_c + 1, 5)];
```

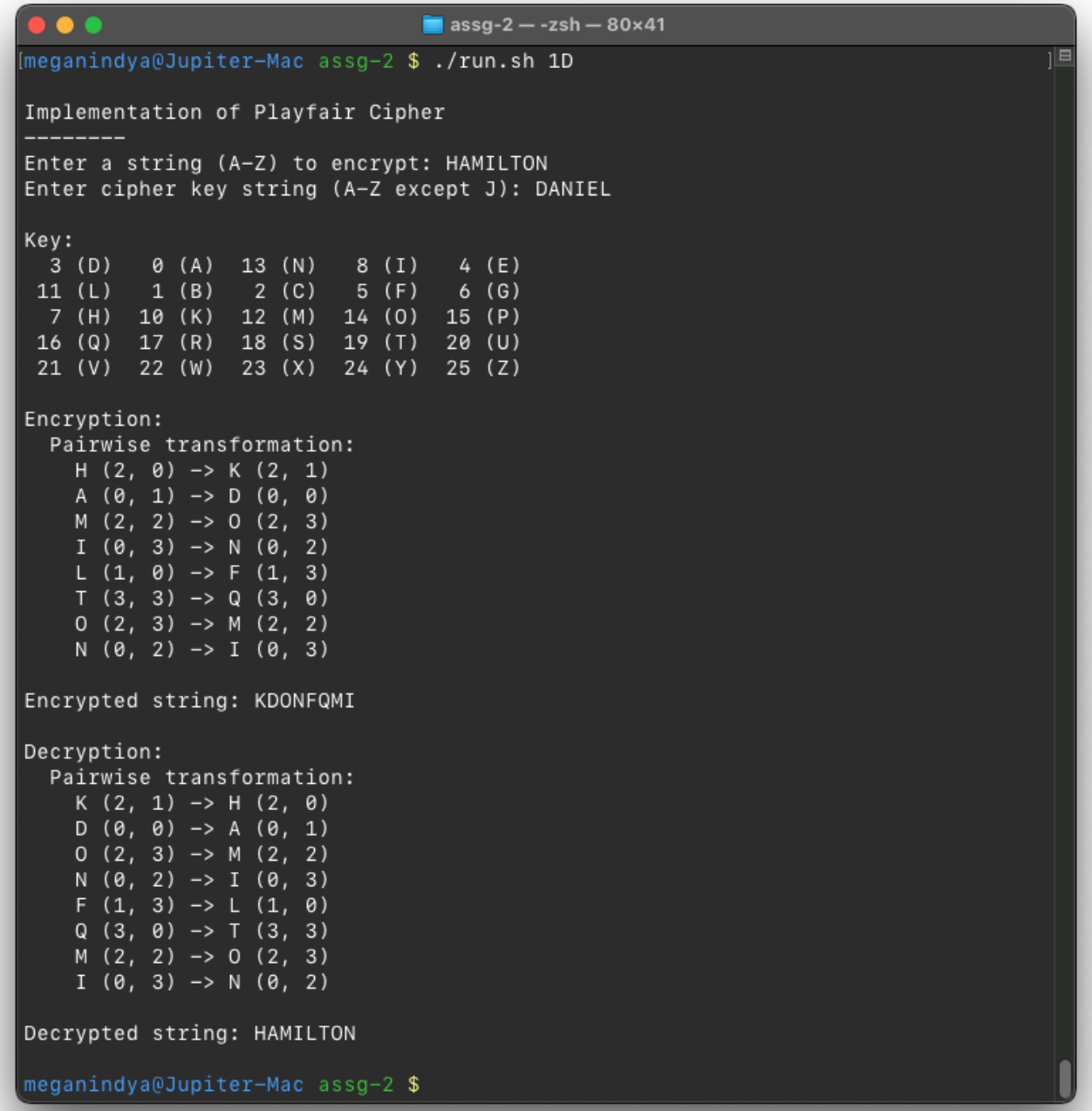
```
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i], a_r, a_c, r_a + 'A', a_r, mod(a_c + 1, 5));
            r_b = key[a_r][mod(b_c + 1, 5)];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i + 1], b_r, b_c, r_b + 'A', a_r, mod(b_c + 1, 5));
        }
        else if (a_c == b_c)
        {
            r_a = key[mod(a_r + 1, 5)][a_c];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i], a_r, a_c, r_a + 'A', mod(a_r + 1, 5), a_c);
            r_b = key[mod(b_r + 1, 5)][a_c];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i + 1], b_r, b_c, r_b + 'A', mod(b_r + 1, 5), a_c);
        }
        else
        {
            r_a = key[a_r][b_c];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i], a_r, a_c, r_a + 'A', a_r, b_c);
            r_b = key[b_r][a_c];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i + 1], b_r, b_c, r_b + 'A', b_r, a_c);
        }

        s[i] = r_a + 'A';
        s[i + 1] = r_b + 'A';
    }
}

/*
 * Cipher playfair decrypts (in place) all characters of a string w.r.t a key.
 *
 * s: the string
 * key: cipher key matrix
 */
void decrypt(char *s, int **key)
{
    int map[26][2];
    map[9][0] = -1; // row of 'J'
    map[9][1] = -1; // col of 'J'
    for (int r = 0; r < 5; r++)
    {
        for (int c = 0; c < 5; c++)
        {
            map[key[r][c]][0] = r;
            map[key[r][c]][1] = c;
        }
    }

    printf("  Pairwise transformation:\n");
    for (int i = 0; s[i] != '\0'; i += 2)
```

```
    {
        int a_r = map[s[i] - 'A'][0];
        int a_c = map[s[i] - 'A'][1];
        int b_r = map[s[i + 1] - 'A'][0];
        int b_c = map[s[i + 1] - 'A'][1];

        int r_a, r_b;

        if (a_r == b_r)
        {
            r_a = key[a_r][mod(a_c - 1, 5)];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i], a_r, a_c, r_a + 'A', a_r, mod(a_c - 1, 5));
            r_b = key[a_r][mod(b_c - 1, 5)];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i + 1], b_r, b_c, r_b + 'A', a_r, mod(b_c - 1, 5));
        }
        else if (a_c == b_c)
        {
            r_a = key[mod(a_r - 1, 5)][a_c];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i], a_r, a_c, r_a + 'A', mod(a_r - 1, 5), a_c);
            r_b = key[mod(b_r - 1, 5)][a_c];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i + 1], b_r, b_c, r_b + 'A', mod(b_r - 1, 5), a_c);
        }
        else
        {
            r_a = key[a_r][b_c];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i], a_r, a_c, r_a + 'A', a_r, b_c);
            r_b = key[b_r][a_c];
            printf("    %c (%d, %d) -
> %c (%d, %d)\n", s[i + 1], b_r, b_c, r_b + 'A', b_r, a_c);
        }

        s[i] = r_a + 'A';
        s[i + 1] = r_b + 'A';
    }
}

// ------------------------------------------------------------------------

int main(int argc, char *argv[])
{
    char s[256], key_s[16];
    printf("\nImplementation of Playfair Cipher\n--------\n");
    int repeat;
    do
    {
        printf("Enter a string (A-Z) to encrypt: ");
        scanf("%s", s);
```

```
        repeat = 0;
        for (int i = 0; s[i] != '\0'; i++)
        {
            if (s[i] < 'A' || s[i] > 'Z')
            {
                printf("  Invalid string, retry\n");
                repeat = 1;
                break;
            }
        }
    } while (repeat);
    do
    {
        printf("Enter cipher key string (A-Z except J): ");
        scanf("%s", key_s);
        repeat = 0;
        for (int i = 0; s[i] != '\0'; i++)
        {
            if (s[i] < 'A' || s[i] > 'Z' || s[i] == 'J')
            {
                printf("  Invalid string, retry\n");
                repeat = 1;
                break;
            }
        }
    } while (repeat);

    int **key;
    key = malloc(5 * sizeof *key);
    for (int i = 0; i < 5; i++)
    {
        key[i] = malloc(5 * sizeof *key[i]);
    }
    int flags[26];
    for (int i = 0; i < 26; i++)
    {
        flags[i] = 0;
    }
    flags[9] = 1;
    for (int i = 0; i < strlen(key_s); i++)
    {
        key[i / 5][i % 5] = key_s[i] - 'A';
        flags[key_s[i] - 'A'] = 1;
    }
    int k = strlen(key_s), ki = 0;
    while (k < 25)
    {
        while (flags[ki] == 1)
        {
            ki++;
        }
        key[k / 5][k % 5] = ki++;
```

```
        k++;
    }

    printf("\nKey:\n");
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            printf("%3d (%c) ", key[i][j], key[i][j] + 'A');
        }
        printf("\n");
    }

    printf("\nEncryption:\n");
    encrypt(s, key);
    printf("\nEncrypted string: %s\n", s);
    printf("\nDecryption:\n");
    decrypt(s, key);
    printf("\nDecrypted string: %s\n", s);
    printf("\n");

    for (int i = 0; i < 5; i++)
    {
        free(key[i]);
    }
    free(key);
}
```

Sample run


assg-2 — -zsh — 80×41
meganindya@Jupiter-Mac assg-2 $ ./run.sh 1D
Implementation of Playfair Cipher
--------
Enter a string (A-Z) to encrypt: HAMILTON
Enter cipher key string (A-Z except J): DANIEL
Key:
3 (D) 0 (A) 13 (N) 8 (I) 4 (E)
11 (L) 1 (B) 2 (C) 5 (F) 6 (G)
7 (H) 10 (K) 12 (M) 14 (O) 15 (P)
16 (Q) 17 (R) 18 (S) 19 (T) 20 (U)
21 (V) 22 (W) 23 (X) 24 (Y) 25 (Z)
Encryption:
Pairwise transformation:
H (2, 0) -> K (2, 1)
A (0, 1) -> D (0, 0)
M (2, 2) -> O (2, 3)
I (0, 3) -> N (0, 2)
L (1, 0) -> F (1, 3)
T (3, 3) -> Q (3, 0)
O (2, 3) -> M (2, 2)
N (0, 2) -> I (0, 3)
Encrypted string: KDONFQMI
Decryption:
Pairwise transformation:
K (2, 1) -> H (2, 0)
D (0, 0) -> A (0, 1)
O (2, 3) -> M (2, 2)
N (0, 2) -> I (0, 3)
F (1, 3) -> L (1, 0)
Q (3, 0) -> T (3, 3)
M (2, 2) -> O (2, 3)
I (0, 3) -> N (0, 2)
Decrypted string: HAMILTON
meganindya@Jupiter-Mac assg-2 $

5. Hill Cipher

Source: `1-e-cipher-hill.c`

```c
#include <stdio.h>  // printf, scanf
#include <stdlib.h> // abs, srand, rand
#include <string.h> // strlen
#include <time.h>   // time

#include "utils.h"
#define MOD 26

/*
 * Cipher hill encrypts (in place) all characters of a string w.r.t a key.
 *
 * s: the string
 * ord_key: order of cipher key matrix
 * key: cipher key matrix
 */
void encrypt(char *s, int ord_key, int **key)
{
    printf("  Original string:\n");
    for (int i = 0; i < strlen(s); i++)
    {
        printf("    %c (%2d)\n", s[i], s[i] - 'A');
    }

    int temp[ord_key];

    printf("\n  Multiplied:\n");
    for (int r = 0; r < ord_key; r++)
    {
        printf("    ");
        for (int c = 0; c < ord_key; c++)
        {
            printf("%3d ", key[r][c]);
        }
        printf("%7d", s[r] - 'A');

        int sum = 0;
        for (int c = 0; c < ord_key; c++)
        {
            sum += key[r][c] * (s[c] - 'A');
        }
        temp[r] = sum;
        printf("   |    %3d\n", sum);
    }

    printf("\n  Multiplied matrix (mod 26):\n");
    for (int i = 0; i < ord_key; i++)
    {
        s[i] = mod_26(temp[i]) + 'A';
        printf("    %c (%2d)\n", s[i], s[i] - 'A');
```

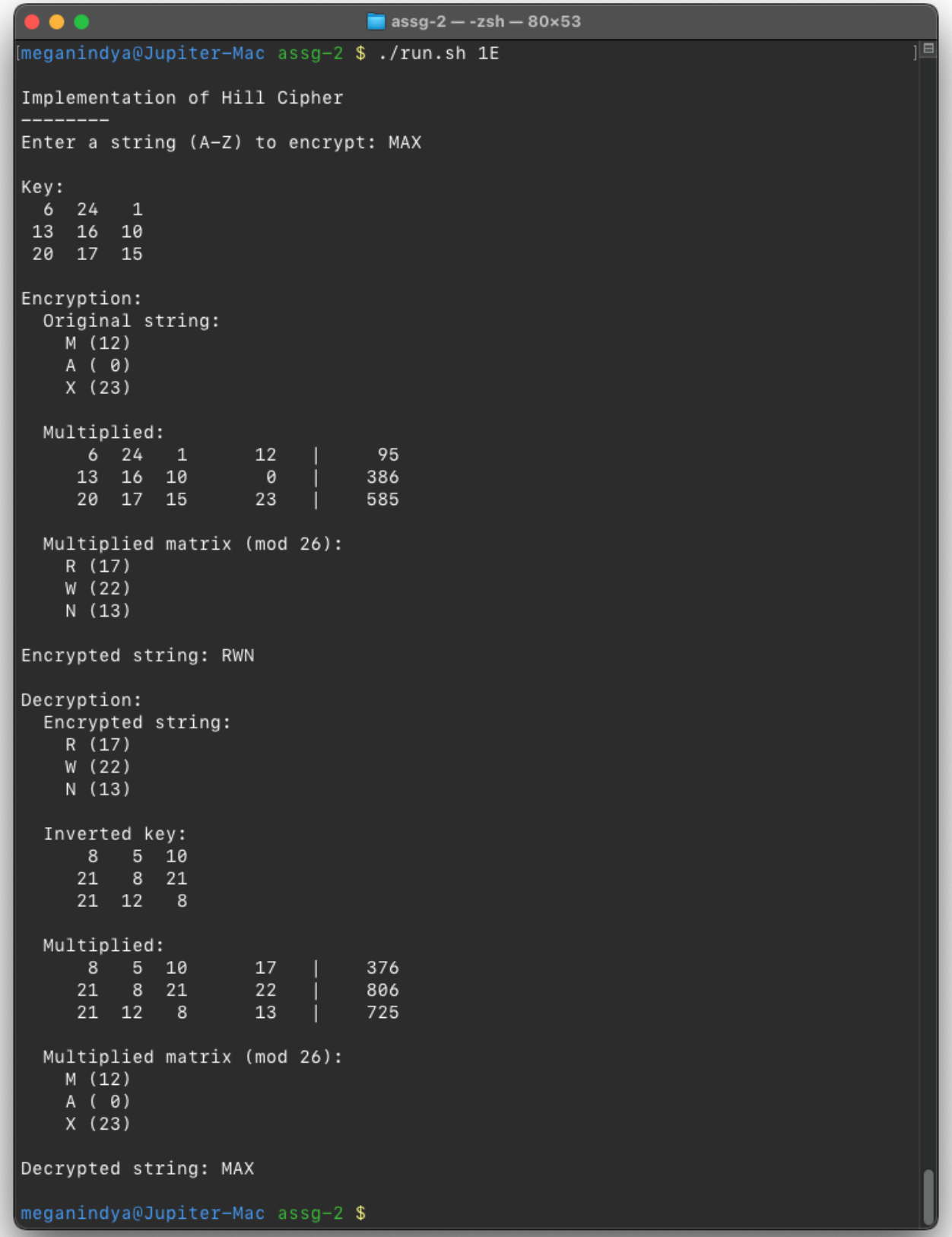
```
    }
}

/*
 * Cipher hill decrypts (in place) all characters of a string w.r.t a key.
 *
 * s: the string
 * ord_key: order of cipher key matrix
 * key: cipher key matrix
 */
void decrypt(char *s, int ord_key, int **key)
{
    printf("  Encrypted string:\n");
    for (int i = 0; i < strlen(s); i++)
    {
        printf("    %c (%2d)\n", s[i], s[i] - 'A');
    }

    int **key_inv;
    key_inv = malloc((ord_key) * sizeof *key_inv);
    for (int i = 0; i < ord_key; i++)
    {
        key_inv[i] = malloc(ord_key * sizeof *key_inv[i]);
    }
    mat_invert(ord_key, key, key_inv);

    printf("\n  Inverted key:\n");
    for (int r = 0; r < ord_key; r++)
    {
        printf("    ");
        for (int c = 0; c < ord_key; c++)
        {
            printf("%3d ", key_inv[r][c]);
        }
        printf("\n");
    }

    int temp[ord_key];

    printf("\n  Multiplied:\n");
    for (int r = 0; r < ord_key; r++)
    {
        printf("    ");
        for (int c = 0; c < ord_key; c++)
        {
            printf("%3d ", key_inv[r][c]);
        }
        printf("%7d", s[r] - 'A');

        int sum = 0;
        for (int c = 0; c < ord_key; c++)
        {
```

```
            sum += key_inv[r][c] * (s[c] - 'A');
        }
        temp[r] = sum;
        printf("   |    %3d\n", sum);
    }

    printf("\n  Multiplied matrix (mod 26):\n");
    for (int i = 0; i < ord_key; i++)
    {
        s[i] = mod_26(temp[i]) + 'A';
        printf("    %c (%2d)\n", s[i], s[i] - 'A');
    }

    for (int i = 0; i < ord_key; i++)
    {
        free(key_inv[i]);
    }
    free(key_inv);
}

// ------------------------------------------------------------------------------

int main(int argc, char *argv[])
{
    char s[256];
    printf("\nImplementation of Hill Cipher\n--------\n");
    int repeat;
    do
    {
        printf("Enter a string (A-Z) to encrypt: ");
        scanf("%s", s);
        repeat = 0;
        for (int i = 0; s[i] != '\0'; i++)
        {
            if (s[i] < 'A' || s[i] > 'Z')
            {
                printf("  Invalid string, retry\n");
                repeat = 1;
                break;
            }
        }
    } while (repeat);

    int ord_key = strlen(s);
    int **key;
    key = malloc((ord_key) * sizeof *key);
    for (int i = 0; i < ord_key; i++)
    {
        key[i] = malloc(ord_key * sizeof *key[i]);
    }

    // Key manually provided for cases 2 and 3 for debugging and reviewing
```

```
    if (ord_key == 2)
    {
        key[0][0] = 25;
        key[0][1] = 22;
        key[1][0] = 4;
        key[1][1] = 21;
    }
    else if (ord_key == 3)
    {
        key[0][0] = 6;
        key[0][1] = 24;
        key[0][2] = 1;
        key[1][0] = 13;
        key[1][1] = 16;
        key[1][2] = 10;
        key[2][0] = 20;
        key[2][1] = 17;
        key[2][2] = 15;
    }
    else
    {
        srand(time(0));
        do
        {
            for (int i = 0; i < ord_key; i++)
            {
                for (int j = 0; j < ord_key; j++)
                {
                    key[i][j] = rand() % 26;
                }
            }

            repeat = mat_invert_check(ord_key, key);
        } while (repeat);
    }

    printf("\nKey:\n");
    for (int i = 0; i < ord_key; i++)
    {
        for (int j = 0; j < ord_key; j++)
        {
            printf("%3d ", key[i][j]);
        }
        printf("\n");
    }

    printf("\nEncryption:\n");
    encrypt(s, ord_key, key);
    printf("\nEncrypted string: %s\n", s);
    printf("\nDecryption:\n");
    decrypt(s, ord_key, key);
    printf("\nDecrypted string: %s\n", s);
```

```
    printf("\n");

    for (int i = 0; i < ord_key; i++)
    {
        free(key[i]);
    }
    free(key);
}
```

Sample run


assg-2 — -zsh — 80×53
meganindya@Jupiter-Mac assg-2 $ ./run.sh 1E
Implementation of Hill Cipher
--------
Enter a string (A-Z) to encrypt: MAX
Key:
6 24 1
13 16 10
20 17 15
Encryption:
Original string:
M (12)
A ( 0)
X (23)
Multiplied:
6 24 1 12 | 95
13 16 10 0 | 386
20 17 15 23 | 585
Multiplied matrix (mod 26):
R (17)
W (22)
N (13)
Encrypted string: RWN
Decryption:
Encrypted string:
R (17)
W (22)
N (13)
Inverted key:
8 5 10
21 8 21
21 12 8
Multiplied:
8 5 10 17 | 376
21 8 21 22 | 806
21 12 8 13 | 725
Multiplied matrix (mod 26):
M (12)
A ( 0)
X (23)
Decrypted string: MAX
meganindya@Jupiter-Mac assg-2 $

2. Write programs to carry out exhaustive key search attacks on the Shift Cipher, Multiplicative Cipher and Affine Cipher that you have implemented. (Aim to attack a cipher is to break its key.)

a. Hence use an exhaustive key search to decrypt the following ciphertext, which was encrypted using a Shift Cipher:

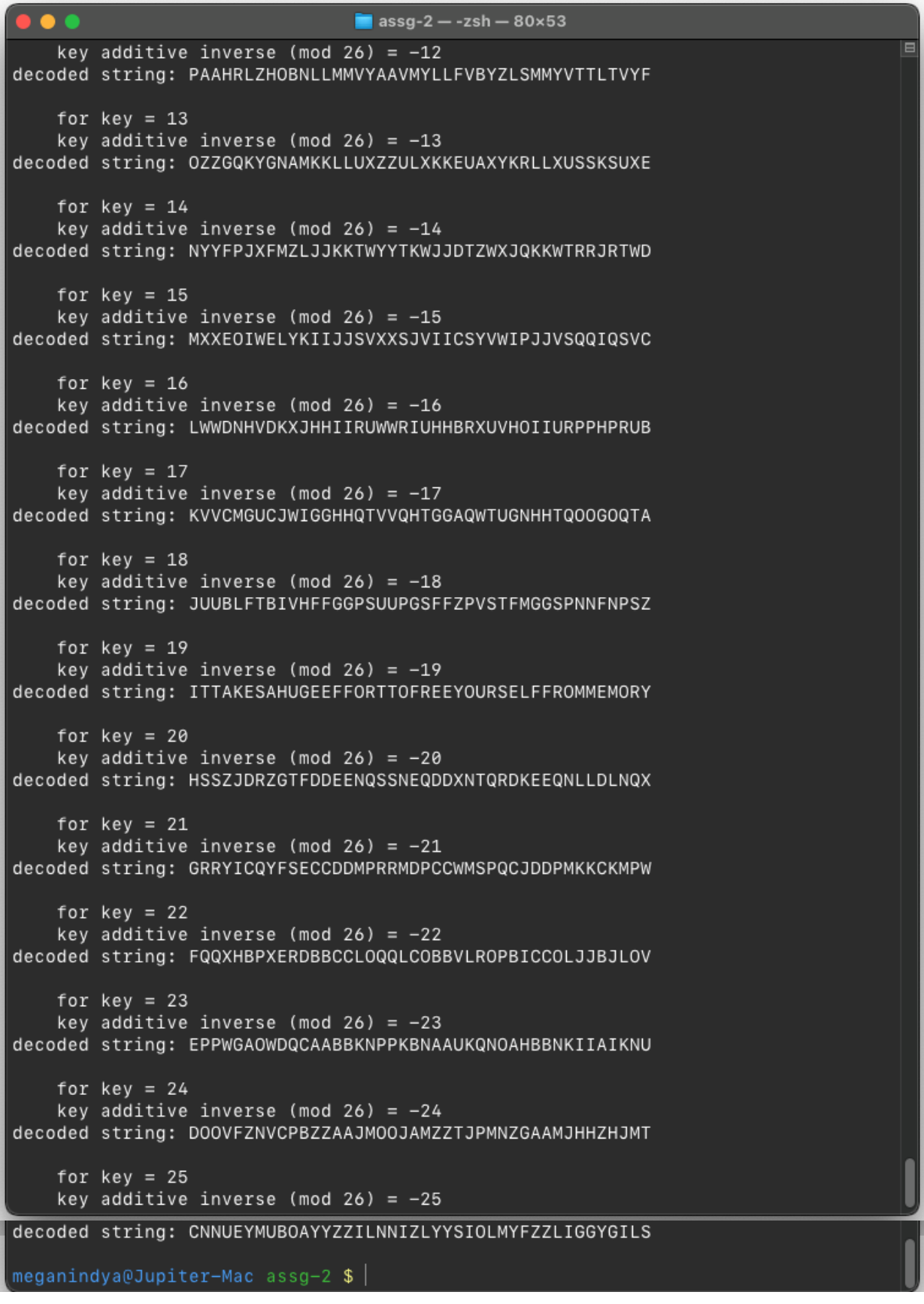
BMMTDXLTANZXXYYHKMMHYKXXRHNKLXEYYKHFFXFHKR

Source: `2-a-attack-cipher-shift.c`

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

#include "utils.h"
#define MOD 26

/*
 * Performs shift cipher decripytion against all additive inverse values in [0, m - 1].
 *
 * s: the encoded string
 */
void attack(char *s)
{
    for (int i = 0; i < MOD; i++)
    {
        int key_inv = mod_26_add_inv(i);
        printf("    for key = %d\n    key additive inverse (mod %d) = %d\n", i, MOD, key_i
nv);

        char enc[64];
        strcpy(enc, s);
        for (int j = 0; enc[j] != '\0'; j++)
        {
            enc[j] = 'A' + mod_26((enc[j] - 'A') + key_inv);
        }
        printf("decoded string: %s\n\n", enc);
    }
}

// ---------------------------------------------------------------------------

int main(int argc, char *argv[])
{
    char s[64] = "BMMTDXLTANZXXYYHKMMHYKXXRHNKLXEYYKHFFXFHKR";
    printf("\nencoded string: %s\n\n", s);
    attack(s);
}
```

Sample run


assg-2 — -zsh — 80×53
meganindya@Jupiter-Mac assg-2 $ ./run.sh 2A
encoded string: BMMTDXLTANZXXYYHKMMHYKXXRHNKLXEYYKHFFXFHKR
for key = 0
key additive inverse (mod 26) = 0
decoded string: BMMTDXLTANZXXYYHKMMHYKXXRHNKLXEYYKHFFXFHKR
for key = 1
key additive inverse (mod 26) = -1
decoded string: ALLSCWKSZMYWWXXGJLLGXJWWQGMJKWDXXJGEEWEGJQ
for key = 2
key additive inverse (mod 26) = -2
decoded string: ZKKRBVJRYLXVVWWFIKKFWIVVPFLIJVCWWIFDDVDFIP
for key = 3
key additive inverse (mod 26) = -3
decoded string: YJJQAUIQXKWUUVVEHJJEVHUUOEKHIUBVVHECCUCEHO
for key = 4
key additive inverse (mod 26) = -4
decoded string: XIIPZTHPWJVTTUUDGIIDUGTTNDJGHTAUUGDBBTBDGN
for key = 5
key additive inverse (mod 26) = -5
decoded string: WHHOYSGOVIUSSTTCFHHCTFSSMCIFGSZTTFCAASACFM
for key = 6
key additive inverse (mod 26) = -6
decoded string: VGGNXRFNUHTRRSSBEGGBSERRLBHEFRYSSEBZZRZBEL
for key = 7
key additive inverse (mod 26) = -7
decoded string: UFFMWQEMTGSQQRRADFFARDQQKAGDEQXRRDAYYQYADK
for key = 8
key additive inverse (mod 26) = -8
decoded string: TEELVPDLSFRPPQQZCEEZQCPPJZFCDPWQQCZXXPXZCJ
for key = 9
key additive inverse (mod 26) = -9
decoded string: SDDKUOCKREQOOPPYBDDYPBOOIYEBCOVPPBYWWOWYBI
for key = 10
key additive inverse (mod 26) = -10
decoded string: RCCJTNBJQDPNNOOXACCXOANNHXDABNUOOAXVVNVXAH
for key = 11
key additive inverse (mod 26) = -11
decoded string: QBBISMAIPCOMMNNWZBBWNZMMGWCZAMTNNZWUUMUWZG
for key = 12


assg-2 — -zsh — 80×53
key additive inverse (mod 26) = -12
decoded string: PAAHRLZHOBNLLMMVYAAVMYLLFVBYZLSMMYVTTLTVYF
for key = 13
key additive inverse (mod 26) = -13
decoded string: OZZGQKYGNAMKKLLUXZZULXKKEUAXYKRLLXUSSKSUXE
for key = 14
key additive inverse (mod 26) = -14
decoded string: NYYFPJXFMZLJJKKTWYYTKWJJDTZWXJQKKWTRRJRTWD
for key = 15
key additive inverse (mod 26) = -15
decoded string: MXXEOIWELYKIIJJSVXXSJVIICSYVWIPJJVSQQIQSVC
for key = 16
key additive inverse (mod 26) = -16
decoded string: LWWDNHVDKXJHHIIRUWWRIUHHBRXUVHOIIURPPHPRUB
for key = 17
key additive inverse (mod 26) = -17
decoded string: KVVCMGUCJWIGGHHQTVVQHTGGAQWTUGNHHTQOOGOQTA
for key = 18
key additive inverse (mod 26) = -18
decoded string: JUUBLFTBIVHFFGGPSUUPGSFFZPVSTFMGGSPNNFNPSZ
for key = 19
key additive inverse (mod 26) = -19
decoded string: ITTAKESAHUGEEFFORTTOFREEYOURSELFFROMMEMORY
for key = 20
key additive inverse (mod 26) = -20
decoded string: HSSZJDRZGTFDDEENQSSNEQDDXNTQRDKEEQNLLDLNQX
for key = 21
key additive inverse (mod 26) = -21
decoded string: GRRYICQYFSECCDDMPRRMDPCCWMSPQCJDDPMKKCKMPW
for key = 22
key additive inverse (mod 26) = -22
decoded string: FQQXHBPXERDBBCCLOQQLCOBBVLROPBICCOLJJBJLOV
for key = 23
key additive inverse (mod 26) = -23
decoded string: EPPWGAOWDQCAABBKNPPKBNAAUKQNOAHBBNKIIAIKNU
for key = 24
key additive inverse (mod 26) = -24
decoded string: DOOVFZNVCPBZZAAJMOOJAMZZTJPMNZGAAMJHHZHJMT
for key = 25
key additive inverse (mod 26) = -25
decoded string: CNNUEYMUBOAYYZZILNNIZLYYSIOLMYFZZLIGGYGILS
meganindya@Jupiter-Mac assg-2 $

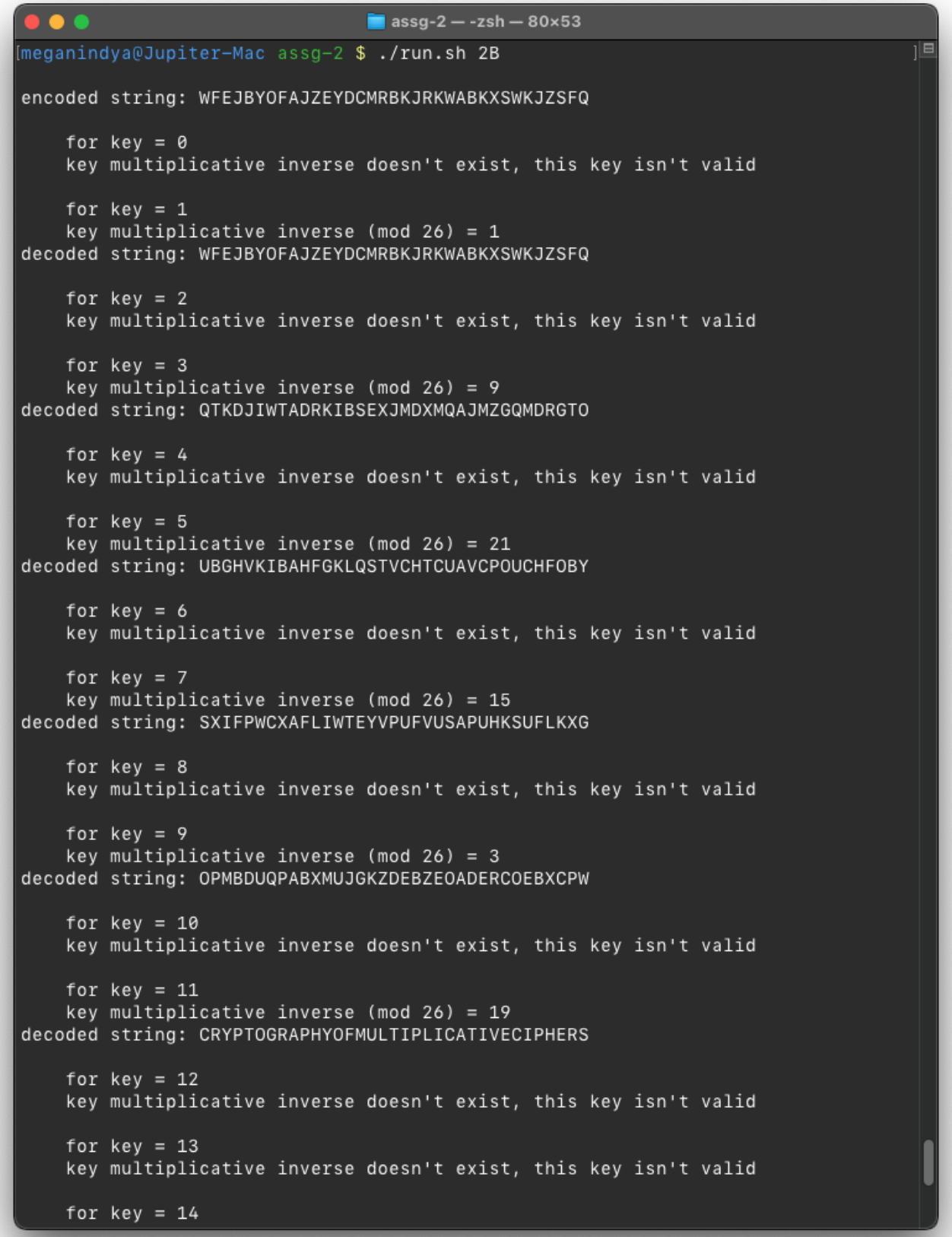
b. Use an exhaustive key search to decrypt the following ciphertext, which was encrypted using a Multiplicative Cipher:

WFEJBYOFAJZEYDCMRBKJRKWABKXSWKJZSFQ

Source: `2-b-attack-cipher-multiplicative.c`

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

#include "utils.h"
#define MOD 26

/*
 * Performs shift cipher decripytion against all valid multiplicative inverse values in [0
, m - 1].
 *
 * s: the encoded string
 */
void attack(char *s)
{
    for (int i = 0; i < MOD; i++)
    {
        int key_inv = mod_26_mul_inv(i);
        if (key_inv == -1)
        {
            printf("    for key = %d\n", i);
            printf("    key multiplicative inverse doesn't exist, this key isn't valid\n\n
");
            continue;
        }
        printf("    for key = %d\n    key multiplicative inverse (mod %d) = %d\n", i, MOD,
 key_inv);

        char enc[64];
        strcpy(enc, s);
        for (int j = 0; enc[j] != '\0'; j++)
        {
            enc[j] = 'A' + mod_26((enc[j] - 'A') * key_inv);
        }
        printf("decoded string: %s\n\n", enc);
    }
}

// ----------------------------------------------------------------------------

int main(int argc, char *argv[])
{
    char s[64] = "WFEJBYOFAJZEYDCMRBKJRKWABKXSWKJZSFQ";
    printf("\nencoded string: %s\n\n", s);
    attack(s);
}
```

Sample run


assg-2 — -zsh — 80×53
meganindya@Jupiter-Mac assg-2 $ ./run.sh 2B
encoded string: WFEJBYOFAJZEYDCMRBKJRKWABKXSWKJZSFQ
for key = 0
key multiplicative inverse doesn't exist, this key isn't valid
for key = 1
key multiplicative inverse (mod 26) = 1
decoded string: WFEJBYOFAJZEYDCMRBKJRKWABKXSWKJZSFQ
for key = 2
key multiplicative inverse doesn't exist, this key isn't valid
for key = 3
key multiplicative inverse (mod 26) = 9
decoded string: QTKDJIWTADRKIBSEXJMDXMQAJMZGQMDRGTO
for key = 4
key multiplicative inverse doesn't exist, this key isn't valid
for key = 5
key multiplicative inverse (mod 26) = 21
decoded string: UBGHVKIBAHFGKLQSTVCHTCUAVCPOUCHFOBY
for key = 6
key multiplicative inverse doesn't exist, this key isn't valid
for key = 7
key multiplicative inverse (mod 26) = 15
decoded string: SXIFPWCXAFLIWTEYVPUFVUSAPUHKSUFLKXG
for key = 8
key multiplicative inverse doesn't exist, this key isn't valid
for key = 9
key multiplicative inverse (mod 26) = 3
decoded string: OPMBDUQPABXMUJGKZDEBZEOADERCOEBXCPW
for key = 10
key multiplicative inverse doesn't exist, this key isn't valid
for key = 11
key multiplicative inverse (mod 26) = 19
decoded string: CRYPTOGRAPHYOFMULTIPLICATIVECIPHERS
for key = 12
key multiplicative inverse doesn't exist, this key isn't valid
for key = 13
key multiplicative inverse doesn't exist, this key isn't valid
for key = 14

```
    key multiplicative inverse doesn't exist, this key isn't valid

    for key = 15
    key multiplicative inverse (mod 26) = 7
decoded string: YJCLHMUJALTCMVOGPHSLPSYAHSFWYSLTWJI

    for key = 16
    key multiplicative inverse doesn't exist, this key isn't valid

    for key = 17
    key multiplicative inverse (mod 26) = 23
decoded string: MLOZXGKLAZDOGRUQBXWZBWMAXWJYMWZDYLE

    for key = 18
    key multiplicative inverse doesn't exist, this key isn't valid

    for key = 19
    key multiplicative inverse (mod 26) = 11
decoded string: IDSVLEYDAVPSEHWCFLGVFGIALGTQIGVPQDU

    for key = 20
    key multiplicative inverse doesn't exist, this key isn't valid

    for key = 21
    key multiplicative inverse (mod 26) = 5
decoded string: GZUTFQSZATVUQPKIHFYTHYGAFYLMGYTVMZC

    for key = 22
    key multiplicative inverse doesn't exist, this key isn't valid

    for key = 23
    key multiplicative inverse (mod 26) = 17
decoded string: KHQXRSEHAXJQSZIWDROXDOKAROBUKOXJUHM

    for key = 24
    key multiplicative inverse doesn't exist, this key isn't valid

    for key = 25
    key multiplicative inverse (mod 26) = 25
decoded string: EVWRZCMVARBWCXYOJZQRJQEAZQDIEQRBIVK

meganindya@Jupiter-Mac assg-2 $
```

c. Use an exhaustive key search to decrypt the following ciphertext, which was encrypted using a Affine Cipher:

EFXECFBDQGGXRADQTFFUFSPGAHQTDGGAFZDJFGHJFBDQGHGDCCGXSFJDHQGAFZDJF

Source: `2-c-attack-cipher-affine.c`

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

#include "utils.h"
#define MOD 26

/*
 * Performs shift cipher decripytion against all valid multiplicative inverse values in [0
, m - 1].
 *
 * s: the encoded string
 */
void attack(char *s)
{
    for (int i = 0; i < MOD; i++)
    {
        int key_mul_inv = mod_26_mul_inv(i);
        if (key_mul_inv == -1)
        {
            printf("for multiplicative key = %d\n", i);
            printf("key multiplicative inverse doesn't exist, this key isn't valid\n\n");
            continue;
        }
        for (int j = 0; j < MOD; j++)
        {
            int key_add_inv = mod_26_add_inv(j);
            printf("    for key (a, b) = (%d, %d)\n", i, j);
            printf("    key (a = %d) multiplicative inverse (mod %d) = %d\n", i, MOD, key_
mul_inv);
            printf("    key (b = %d) additive inverse (mod %d) = %d\n", j, MOD, key_add_in
v);

            char enc[80];
            strcpy(enc, s);
            for (int x = 0; enc[x] != '\0'; x++)
            {
                enc[x] = 'A' + mod_26(((enc[x] - 'A') + key_add_inv) * key_mul_inv);
            }
            printf("decoded string: %s\n\n", enc);
        }
    }
}

// -----------------------------------------------------------------------------

int main(int argc, char *argv[])
```

```
{
    char s[80] = "EFXECFBDQGGXRADQTFFUFSPGAHQTDGGAFZDJFGHJFBDQGHGDCCGXSFJDHQGAFZDJF";
    printf("\nencoded string: %s\n\n", s);
    attack(s);
}
```

Sample run

```
    for key (a, b) = (7, 3)
    key (a = 7) multiplicative inverse (mod 26) = 15
    key (b = 3) additive inverse (mod 26) = -3
decoded string: PEOPLEWANTTOCHANGEEVERYTHINGATTHESAMETIMEWANTITALLTOREMAINTHESAME
```