Assignment 3

IT851: Information and Systems Security Lab

ANINDYA KUNDU

IT, 8th Semester

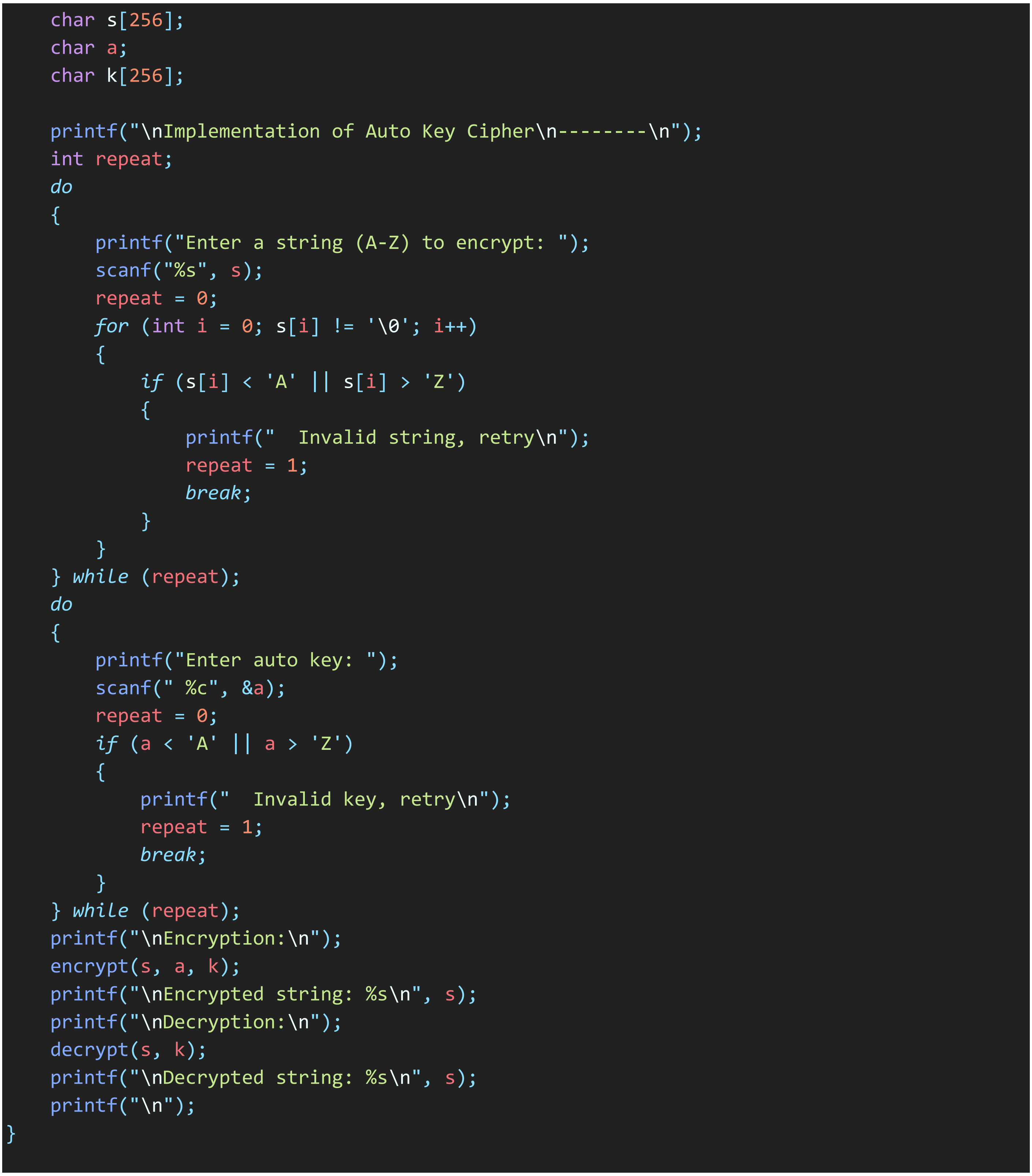
ID 510817020

Repository:
github.com/meganindya/btech-assignments/information-and-systems-security/assg-3

1. Implement the Auto-key Cipher

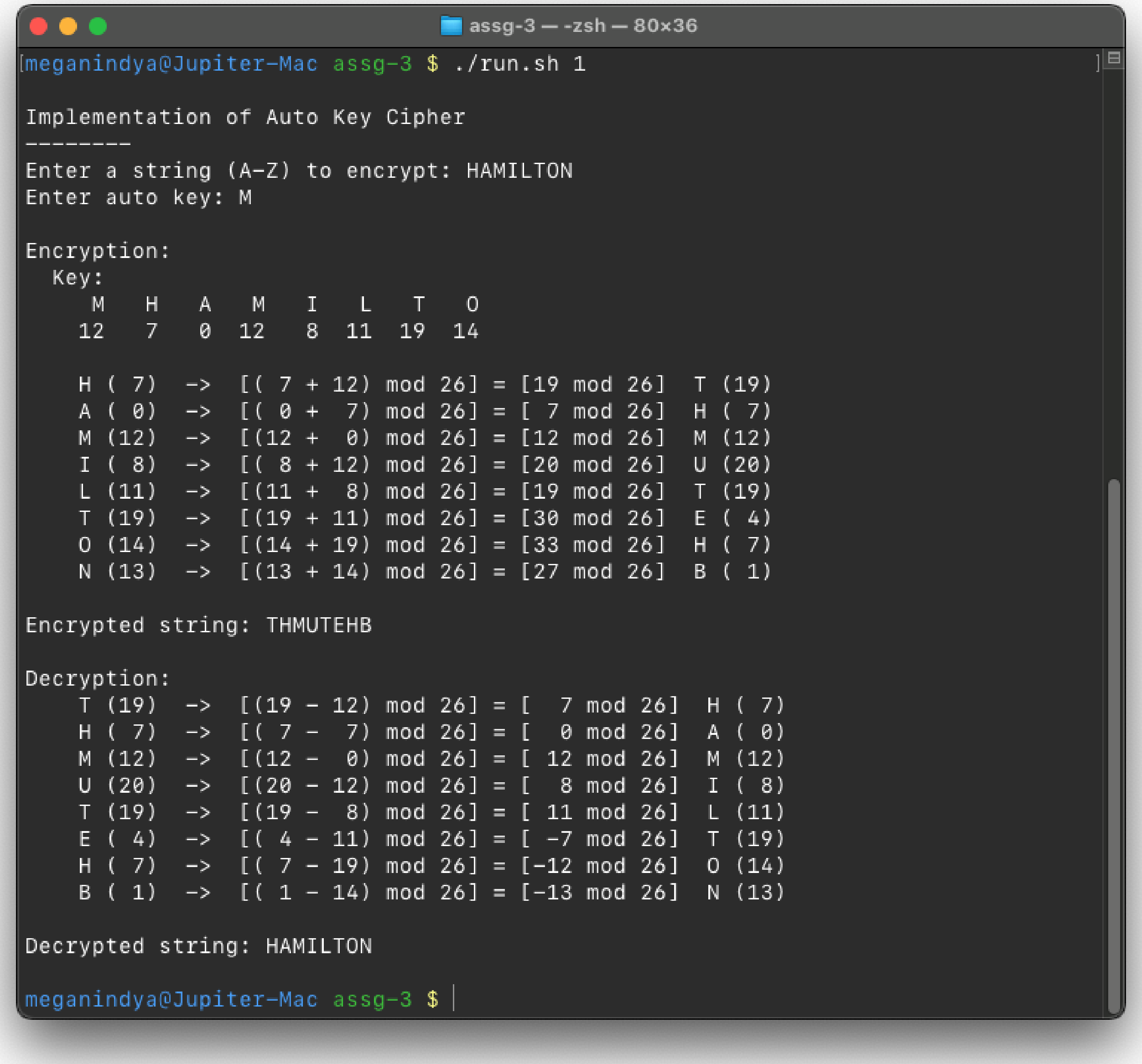
Source: `1-cipher-autokey.c`

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

#include "../utils.h"
#define MOD 26

/*
 * Cipher Auto Key encrypts (in place) all input string and generates a key string.
 *
 * s: the string
 * a: auto key
 * k: array to fill generated key in
 */
void encrypt(char *s, char a, char *k)
{
    int len = strlen(s);

    k[0] = a;
    for (int i = 1; i < len; i++)
    {
        k[i] = s[i - 1];
    }
    k[len] = '\0';

    printf("  Key:\n");
    printf("    ");
    for (int i = 0; i < len; i++)
    {
        printf(" %c  ", k[i]);
    }
    printf("\n    ");
    for (int i = 0; i < len; i++)
    {
        printf("%2d  ", k[i] - 'A');
    }
    printf("\n\n");

    for (int i = 0; i < len; i++)
    {
        char c = s[i];
        int c_n = c - 'A';
        char r = k[i];
        int r_n = r - 'A';
        int add_n = c_n + r_n;
        int enc_n = mod_26(add_n);
        char enc = enc_n + 'A';
        printf(
            "    %c (%2d)  ->  [(%2d + %2d) mod %d] = [%2d mod %d]  %c (%2d)\n",
            c,
            c_n,
            c_n,
            r_n,
            MOD,
            add_n,
            MOD,
            enc,
            enc_n);
        s[i] = enc;
    }
}

/*
 * Cipher Auto Key decrypts (in place) all characters of a string.
 *
 * s: the string
 * k: key string
 */
void decrypt(char *s, char *k)
{
    int len = strlen(k);

    for (int i = 0; i < len; i++)
    {
        char c = s[i];
        int c_n = c - 'A';
        char r = k[i];
        int r_n = r - 'A';
        int sub_n = c_n - r_n;
        int enc_n = mod_26(sub_n);
        char enc = enc_n + 'A';
        printf(
            "    %c (%2d)  ->  [(%2d - %2d) mod %d] = [%3d mod %d]  %c (%2d)\n",
            c,
            c_n,
            c_n,
            r_n,
            MOD,
            sub_n,
            MOD,
            enc,
            enc_n);
        s[i] = enc;
    }
}

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

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



```c
    char s[256];
    char a;
    char k[256];

    printf("\nImplementation of Auto Key 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 auto key: ");
        scanf(" %c", &a);
        repeat = 0;
        if (a < 'A' || a > 'Z')
        {
            printf("  Invalid key, retry\n");
            repeat = 1;
            break;
        }
    } while (repeat);
    printf("\nEncryption:\n");
    encrypt(s, a, k);
    printf("\nEncrypted string: %s\n", s);
    printf("\nDecryption:\n");
    decrypt(s, k);
    printf("\nDecrypted string: %s\n", s);
    printf("\n");
}
```

Sample run



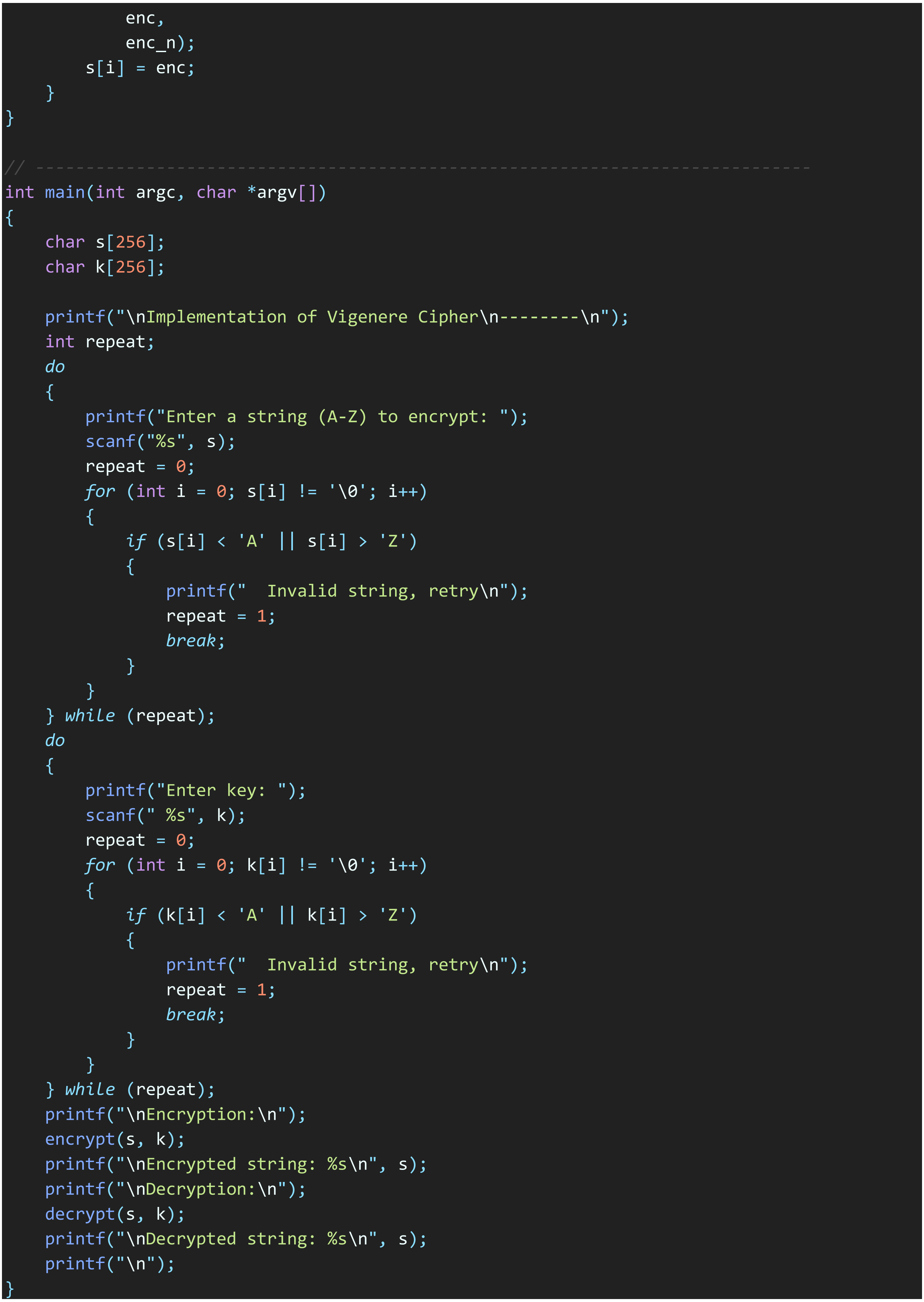
```
meganindya@Jupiter-Mac assg-3 $ ./run.sh 1

Implementation of Auto Key Cipher
--------
Enter a string (A-Z) to encrypt: HAMILTON
Enter auto key: M

Encryption:
  Key:
     M   H   A   M   I   L   T   O
    12   7   0  12   8  11  19  14

    H ( 7)  ->  [( 7 + 12) mod 26] = [19 mod 26]  T (19)
    A ( 0)  ->  [( 0 +  7) mod 26] = [ 7 mod 26]  H ( 7)
    M (12)  ->  [(12 +  0) mod 26] = [12 mod 26]  M (12)
    I ( 8)  ->  [( 8 + 12) mod 26] = [20 mod 26]  U (20)
    L (11)  ->  [(11 +  8) mod 26] = [19 mod 26]  T (19)
    T (19)  ->  [(19 + 11) mod 26] = [30 mod 26]  E ( 4)
    O (14)  ->  [(14 + 19) mod 26] = [33 mod 26]  H ( 7)
    N (13)  ->  [(13 + 14) mod 26] = [27 mod 26]  B ( 1)

Encrypted string: THMUTEHB

Decryption:
    T (19)  ->  [(19 - 12) mod 26] = [  7 mod 26]  H ( 7)
    H ( 7)  ->  [( 7 -  7) mod 26] = [  0 mod 26]  A ( 0)
    M (12)  ->  [(12 -  0) mod 26] = [ 12 mod 26]  M (12)
    U (20)  ->  [(20 - 12) mod 26] = [  8 mod 26]  I ( 8)
    T (19)  ->  [(19 -  8) mod 26] = [ 11 mod 26]  L (11)
    E ( 4)  ->  [( 4 - 11) mod 26] = [ -7 mod 26]  T (19)
    H ( 7)  ->  [( 7 - 19) mod 26] = [-12 mod 26]  O (14)
    B ( 1)  ->  [( 1 - 14) mod 26] = [-13 mod 26]  N (13)

Decrypted string: HAMILTON

meganindya@Jupiter-Mac assg-3 $ 
```

2. Implement the following classic polyalphabetic ciphers (Generate the keys pseudo-randomly, check validity of the key, and store it into a key-file):

a. Vigenere Cipher

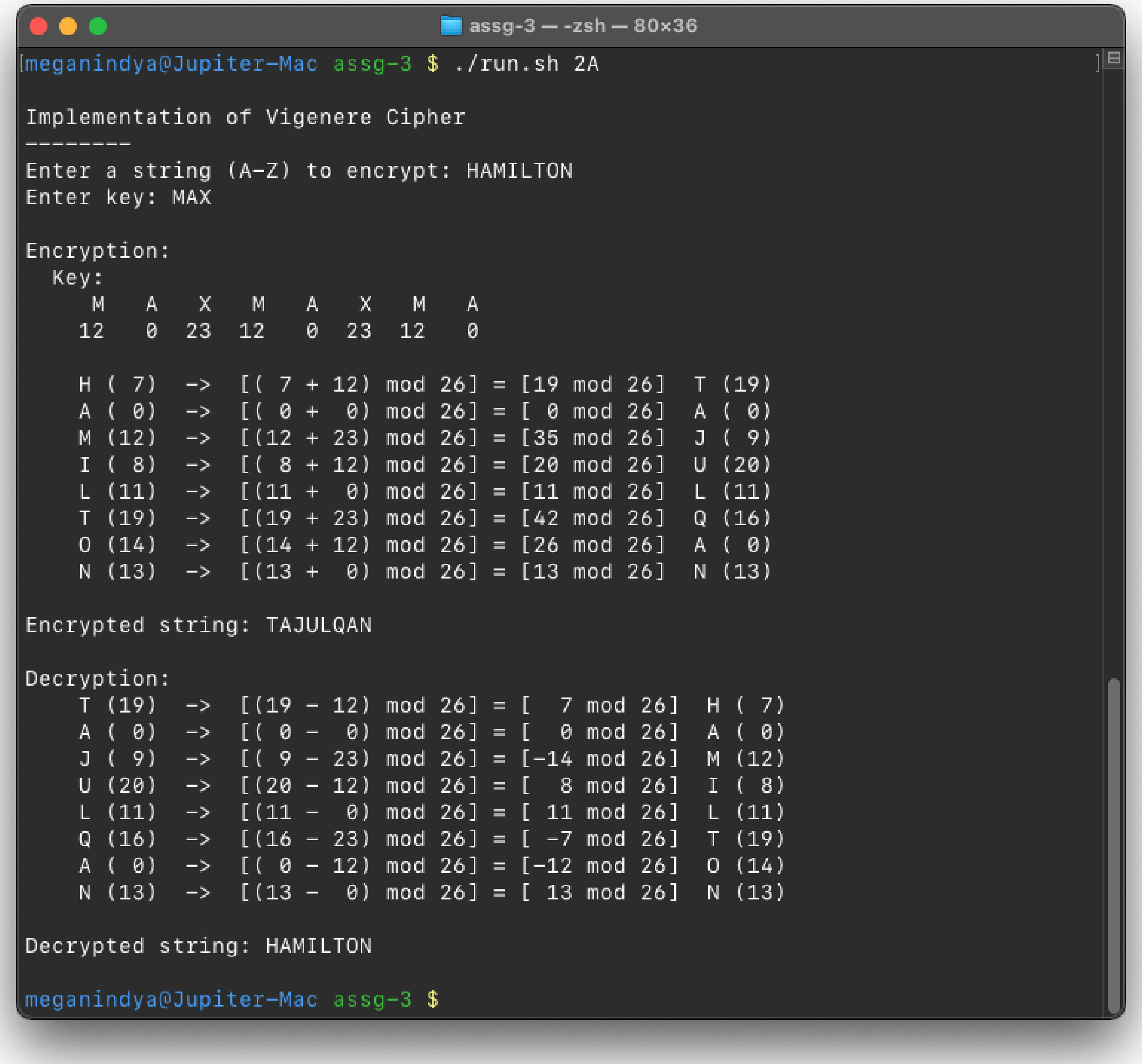
Source: `2-a-cipher-vigenere.c`

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

#include "../utils.h"
#define MOD 26

/*
 * Cipher Vigenere encrypts (in place) all input string and generates a key string.
 *
 * s: the string
 * key: key string
 */
void encrypt(char *s, char *key)
{
    int len = strlen(s);

    int len_k = strlen(key);
    char k[256];
    for (int i = 0; i < len; i++)
    {
        k[i] = key[i % len_k];
    }
    k[len] = '\0';

    printf("  Key:\n");
    printf("    ");
    for (int i = 0; i < len; i++)
    {
        printf(" %c  ", k[i]);
    }
    printf("\n    ");
    for (int i = 0; i < len; i++)
    {
        printf("%2d  ", k[i] - 'A');
    }
    printf("\n\n");

    for (int i = 0; i < len; i++)
    {
        char c = s[i];
        int c_n = c - 'A';
        char r = k[i];
        int r_n = r - 'A';
        int add_n = c_n + r_n;
        int enc_n = mod_26(add_n);
        char enc = enc_n + 'A';
        printf(
            "    %c (%2d)  ->  [(%2d + %2d) mod %d] = [%2d mod %d]  %c (%2d)\n",
            c,
            c_n,
            c_n,
            r_n,
            MOD,
            add_n,
            MOD,
            enc,
            enc_n);
        s[i] = enc;
    }
}

/*
 * Cipher Vigenere decrypts (in place) all characters of a string.
 *
 * s: the string
 * key: key string
 */
void decrypt(char *s, char *key)
{
    int len = strlen(s);

    int len_k = strlen(key);
    char k[256];
    for (int i = 0; i < len; i++)
    {
        k[i] = key[i % len_k];
    }
    k[len] = '\0';

    for (int i = 0; i < len; i++)
    {
        char c = s[i];
        int c_n = c - 'A';
        char r = k[i];
        int r_n = r - 'A';
        int sub_n = c_n - r_n;
        int enc_n = mod_26(sub_n);
        char enc = enc_n + 'A';
        printf(
            "    %c (%2d)  ->  [(%2d - %2d) mod %d] = [%3d mod %d]  %c (%2d)\n",
            c,
            c_n,
            c_n,
            r_n,
            MOD,
            sub_n,
            MOD,
```



```
            enc,
            enc_n);
        s[i] = enc;
    }
}

// -----------------------------------------------------------------------------
int main(int argc, char *argv[])
{
    char s[256];
    char k[256];

    printf("\nImplementation of Vigenere 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 key: ");
        scanf(" %s", k);
        repeat = 0;
        for (int i = 0; k[i] != '\0'; i++)
        {
            if (k[i] < 'A' || k[i] > 'Z')
            {
                printf("  Invalid string, retry\n");
                repeat = 1;
                break;
            }
        }
    } while (repeat);
    printf("\nEncryption:\n");
    encrypt(s, k);
    printf("\nEncrypted string: %s\n", s);
    printf("\nDecryption:\n");
    decrypt(s, k);
    printf("\nDecrypted string: %s\n", s);
    printf("\n");
}
```

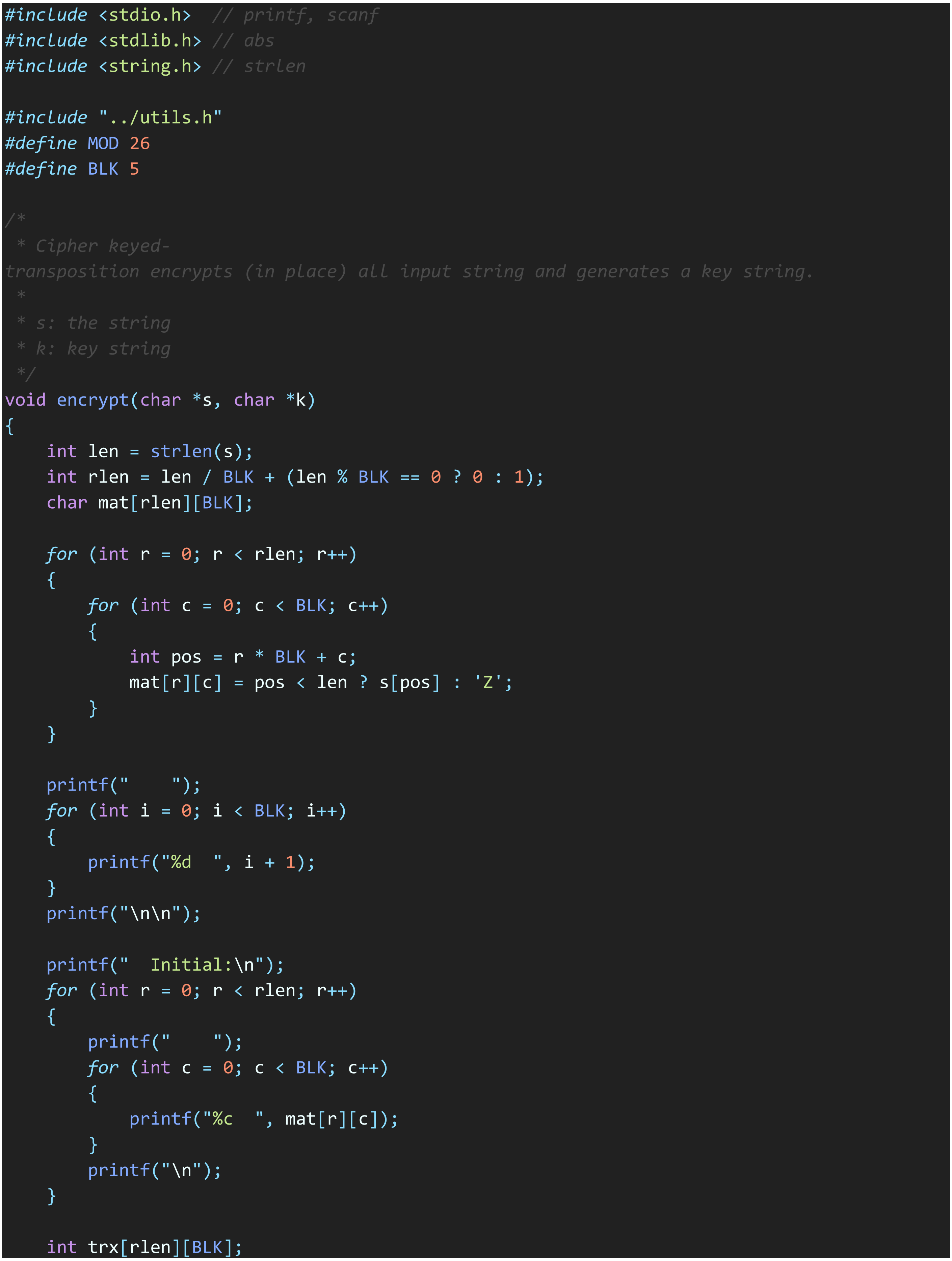
Sample run



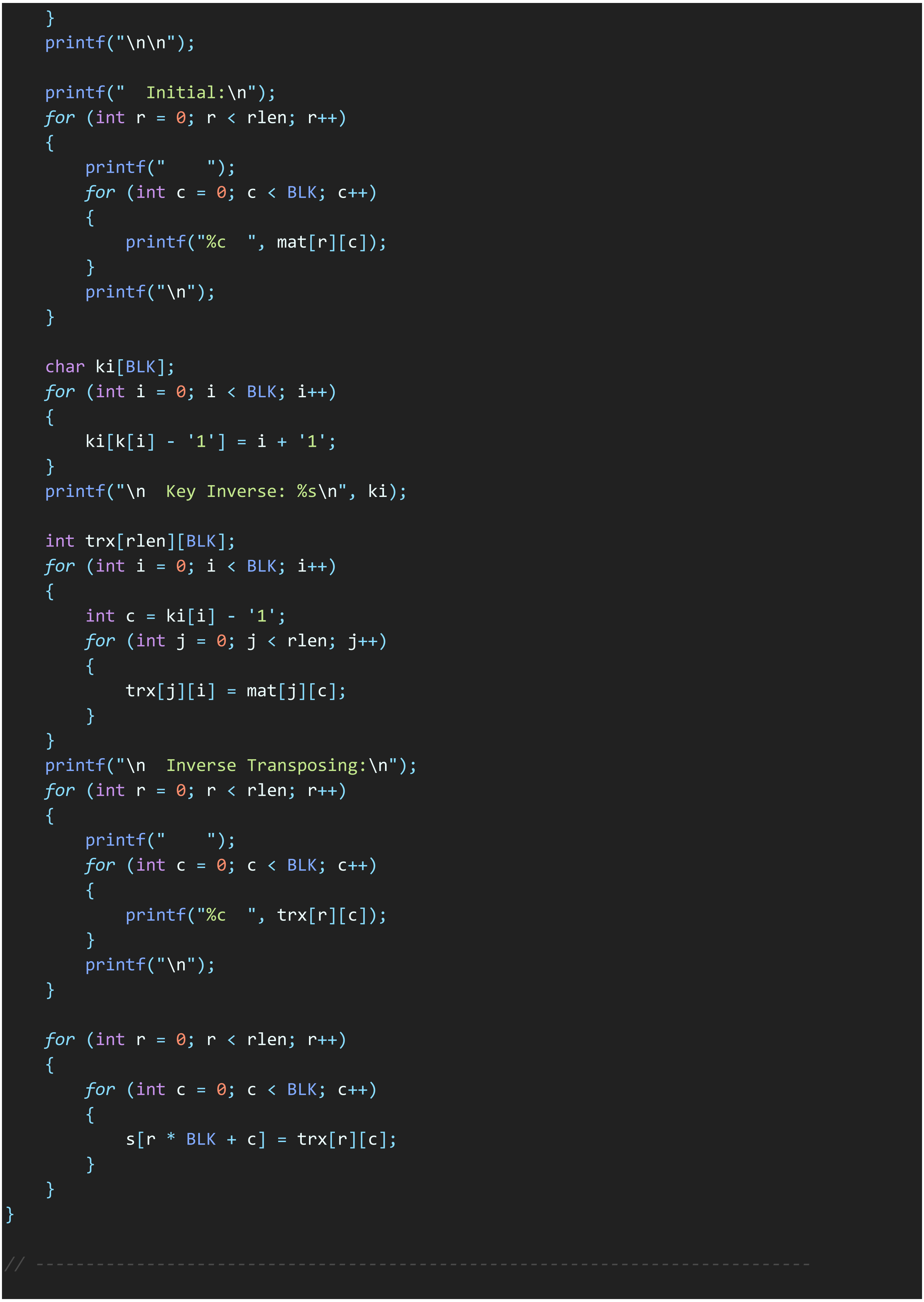
```
[meganindya@Jupiter-Mac assg-3 $ ./run.sh 2A

Implementation of Vigenere Cipher
--------
Enter a string (A-Z) to encrypt: HAMILTON
Enter key: MAX

Encryption:
  Key:
     M   A   X   M   A   X   M   A
    12   0  23  12   0  23  12   0

    H ( 7)  ->  [( 7 + 12) mod 26] = [19 mod 26]  T (19)
    A ( 0)  ->  [( 0 +  0) mod 26] = [ 0 mod 26]  A ( 0)
    M (12)  ->  [(12 + 23) mod 26] = [35 mod 26]  J ( 9)
    I ( 8)  ->  [( 8 + 12) mod 26] = [20 mod 26]  U (20)
    L (11)  ->  [(11 +  0) mod 26] = [11 mod 26]  L (11)
    T (19)  ->  [(19 + 23) mod 26] = [42 mod 26]  Q (16)
    O (14)  ->  [(14 + 12) mod 26] = [26 mod 26]  A ( 0)
    N (13)  ->  [(13 +  0) mod 26] = [13 mod 26]  N (13)

Encrypted string: TAJULQAN

Decryption:
    T (19)  ->  [(19 - 12) mod 26] = [  7 mod 26]  H ( 7)
    A ( 0)  ->  [( 0 -  0) mod 26] = [  0 mod 26]  A ( 0)
    J ( 9)  ->  [( 9 - 23) mod 26] = [-14 mod 26]  M (12)
    U (20)  ->  [(20 - 12) mod 26] = [  8 mod 26]  I ( 8)
    L (11)  ->  [(11 -  0) mod 26] = [ 11 mod 26]  L (11)
    Q (16)  ->  [(16 - 23) mod 26] = [ -7 mod 26]  T (19)
    A ( 0)  ->  [( 0 - 12) mod 26] = [-12 mod 26]  O (14)
    N (13)  ->  [(13 -  0) mod 26] = [ 13 mod 26]  N (13)

Decrypted string: HAMILTON

meganindya@Jupiter-Mac assg-3 $
```

b. Keyed Transposition Cipher (assume block size to be 5)

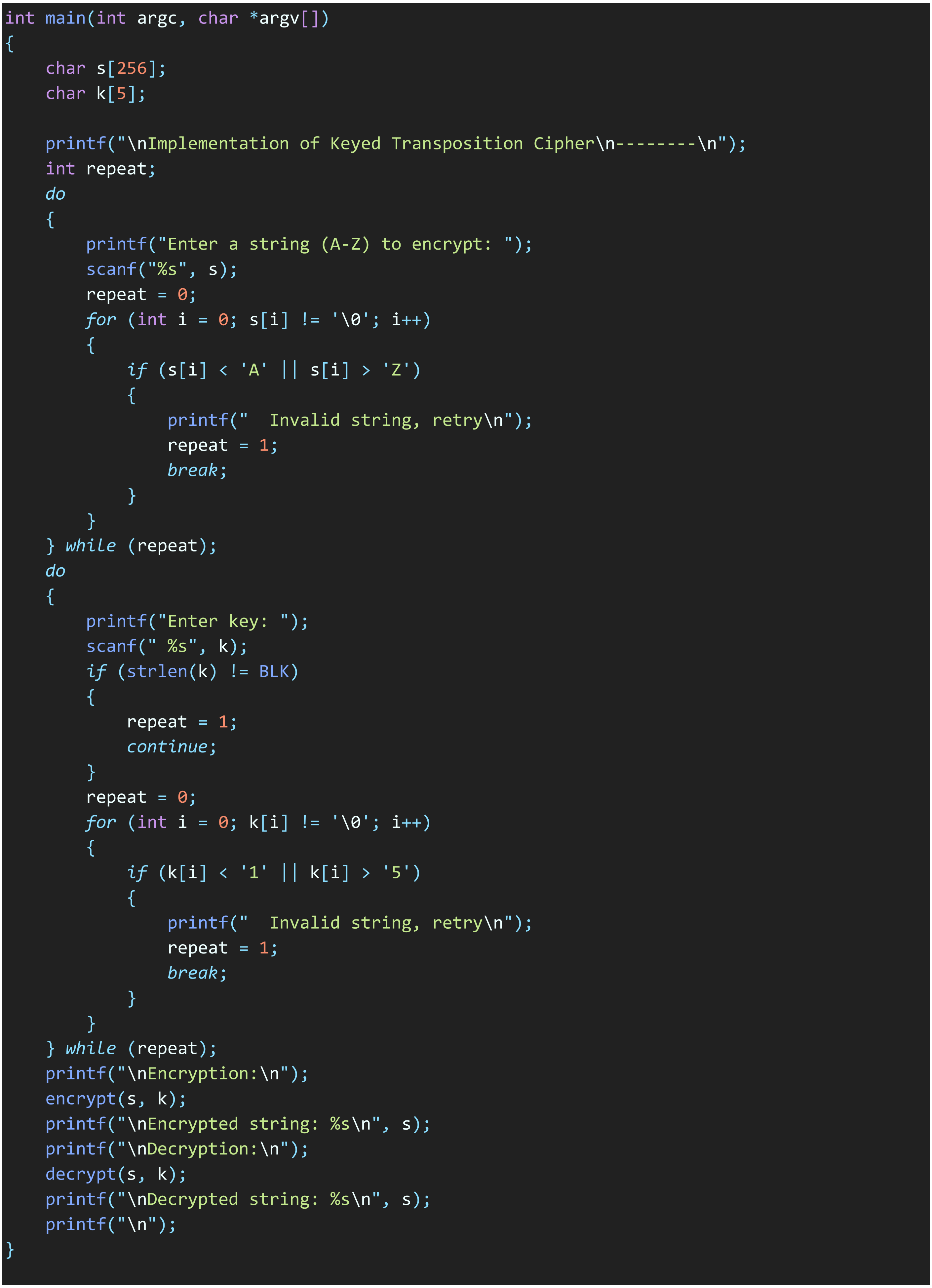
Source: `2-b-cipher-keyed-transposition.c`



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

#include "../utils.h"
#define MOD 26
#define BLK 5

/*
 * Cipher keyed-transposition encrypts (in place) all input string and generates a key string.
 *
 * s: the string
 * k: key string
 */
void encrypt(char *s, char *k)
{
    int len = strlen(s);
    int rlen = len / BLK + (len % BLK == 0 ? 0 : 1);
    char mat[rlen][BLK];

    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < BLK; c++)
        {
            int pos = r * BLK + c;
            mat[r][c] = pos < len ? s[pos] : 'Z';
        }
    }

    printf("    ");
    for (int i = 0; i < BLK; i++)
    {
        printf("%d  ", i + 1);
    }
    printf("\n\n");

    printf("  Initial:\n");
    for (int r = 0; r < rlen; r++)
    {
        printf("    ");
        for (int c = 0; c < BLK; c++)
        {
            printf("%c  ", mat[r][c]);
        }
        printf("\n");
    }

    int trx[rlen][BLK];
    for (int i = 0; i < BLK; i++)
    {
        int c = k[i] - '1';
        for (int j = 0; j < rlen; j++)
        {
            trx[j][i] = mat[j][c];
        }
    }
    printf("\n  Transposing:\n");
    for (int r = 0; r < rlen; r++)
    {
        printf("    ");
        for (int c = 0; c < BLK; c++)
        {
            printf("%c  ", trx[r][c]);
        }
        printf("\n");
    }

    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < BLK; c++)
        {
            s[r * BLK + c] = trx[r][c];
        }
    }
}

/*
 * Cipher keyed-transposition decrypts (in place) all characters of a string.
 *
 * s: the string
 * k: key string
 */
void decrypt(char *s, char *k)
{
    int len = strlen(s);
    int rlen = len / BLK + (len % BLK == 0 ? 0 : 1);
    char mat[rlen][BLK];

    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < BLK; c++)
        {
            mat[r][c] = s[r * BLK + c];
        }
    }

    printf("    ");
    for (int i = 0; i < BLK; i++)
    {
        printf("%d  ", i + 1);
```

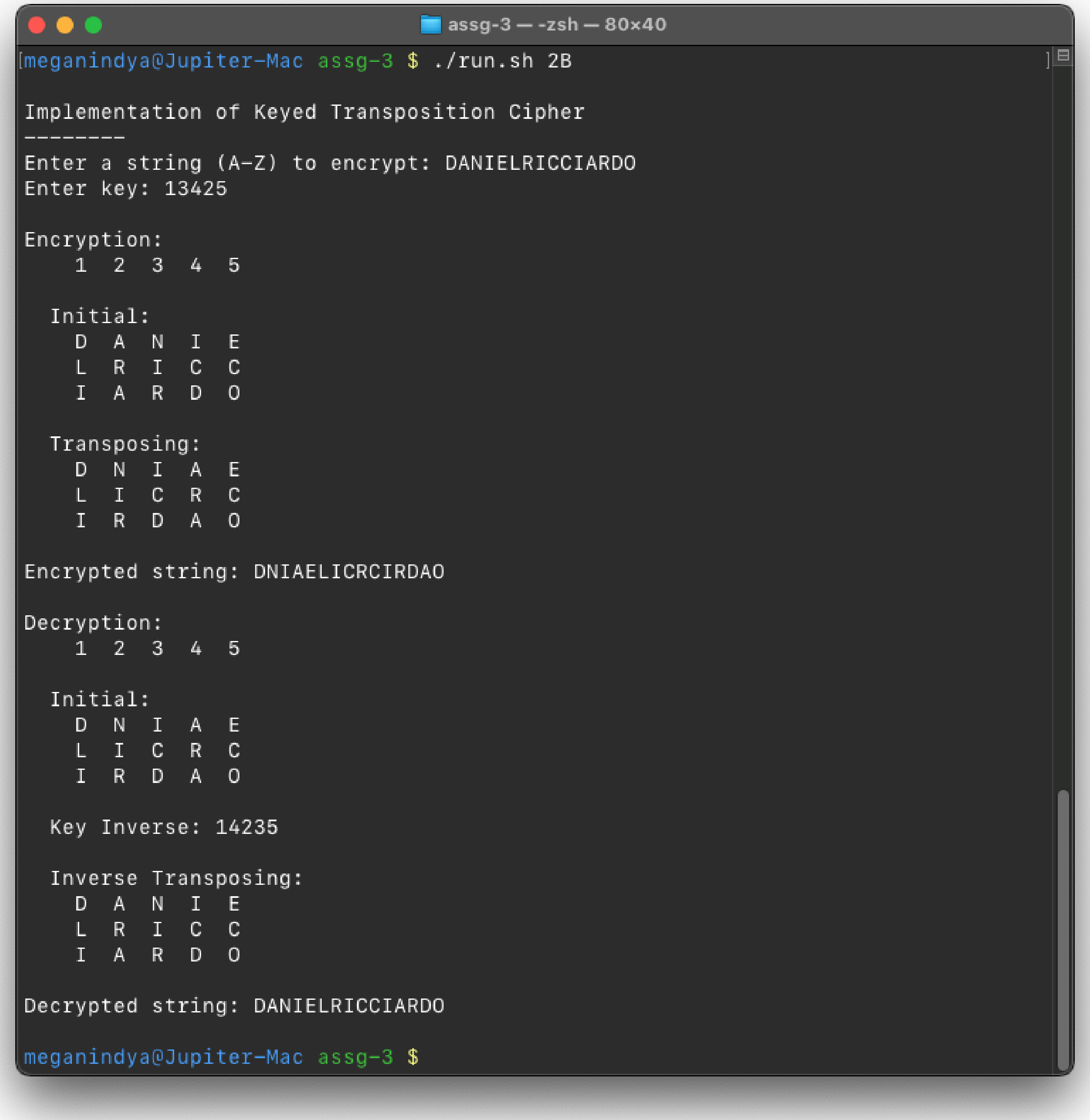


```
    }
    printf("\n\n");

    printf("  Initial:\n");
    for (int r = 0; r < rlen; r++)
    {
        printf("    ");
        for (int c = 0; c < BLK; c++)
        {
            printf("%c  ", mat[r][c]);
        }
        printf("\n");
    }

    char ki[BLK];
    for (int i = 0; i < BLK; i++)
    {
        ki[k[i] - '1'] = i + '1';
    }
    printf("\n  Key Inverse: %s\n", ki);

    int trx[rlen][BLK];
    for (int i = 0; i < BLK; i++)
    {
        int c = ki[i] - '1';
        for (int j = 0; j < rlen; j++)
        {
            trx[j][i] = mat[j][c];
        }
    }
    printf("\n  Inverse Transposing:\n");
    for (int r = 0; r < rlen; r++)
    {
        printf("    ");
        for (int c = 0; c < BLK; c++)
        {
            printf("%c  ", trx[r][c]);
        }
        printf("\n");
    }

    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < BLK; c++)
        {
            s[r * BLK + c] = trx[r][c];
        }
    }
}

// ----------------------------------------------------------------------------
```



```c
int main(int argc, char *argv[])
{
    char s[256];
    char k[5];

    printf("\nImplementation of Keyed Transposition 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 key: ");
        scanf(" %s", k);
        if (strlen(k) != BLK)
        {
            repeat = 1;
            continue;
        }
        repeat = 0;
        for (int i = 0; k[i] != '\0'; i++)
        {
            if (k[i] < '1' || k[i] > '5')
            {
                printf("  Invalid string, retry\n");
                repeat = 1;
                break;
            }
        }
    } while (repeat);
    printf("\nEncryption:\n");
    encrypt(s, k);
    printf("\nEncrypted string: %s\n", s);
    printf("\nDecryption:\n");
    decrypt(s, k);
    printf("\nDecrypted string: %s\n", s);
    printf("\n");
}
```

Sample run



```
meganindya@Jupiter-Mac assg-3 $ ./run.sh 2B

Implementation of Keyed Transposition Cipher
--------
Enter a string (A-Z) to encrypt: DANIELRICCIARDO
Enter key: 13425

Encryption:
    1  2  3  4  5

  Initial:
    D  A  N  I  E
    L  R  I  C  C
    I  A  R  D  O

  Transposing:
    D  N  I  A  E
    L  I  C  R  C
    I  R  D  A  O

Encrypted string: DNIAELICRCIRDAO

Decryption:
    1  2  3  4  5

  Initial:
    D  N  I  A  E
    L  I  C  R  C
    I  R  D  A  O

  Key Inverse: 14235

  Inverse Transposing:
    D  A  N  I  E
    L  R  I  C  C
    I  A  R  D  O

Decrypted string: DANIELRICCIARDO

meganindya@Jupiter-Mac assg-3 $
```

3. Modify the Hill Cipher program you wrote for Assignment 1, so as to implement the following Permutation Cipher: (Following π is a permutation of plain text letters positioned at {1, …, 8})

x	1	2	3	4	5	6	7	8
π (x)	4	1	6	2	7	3	8	5

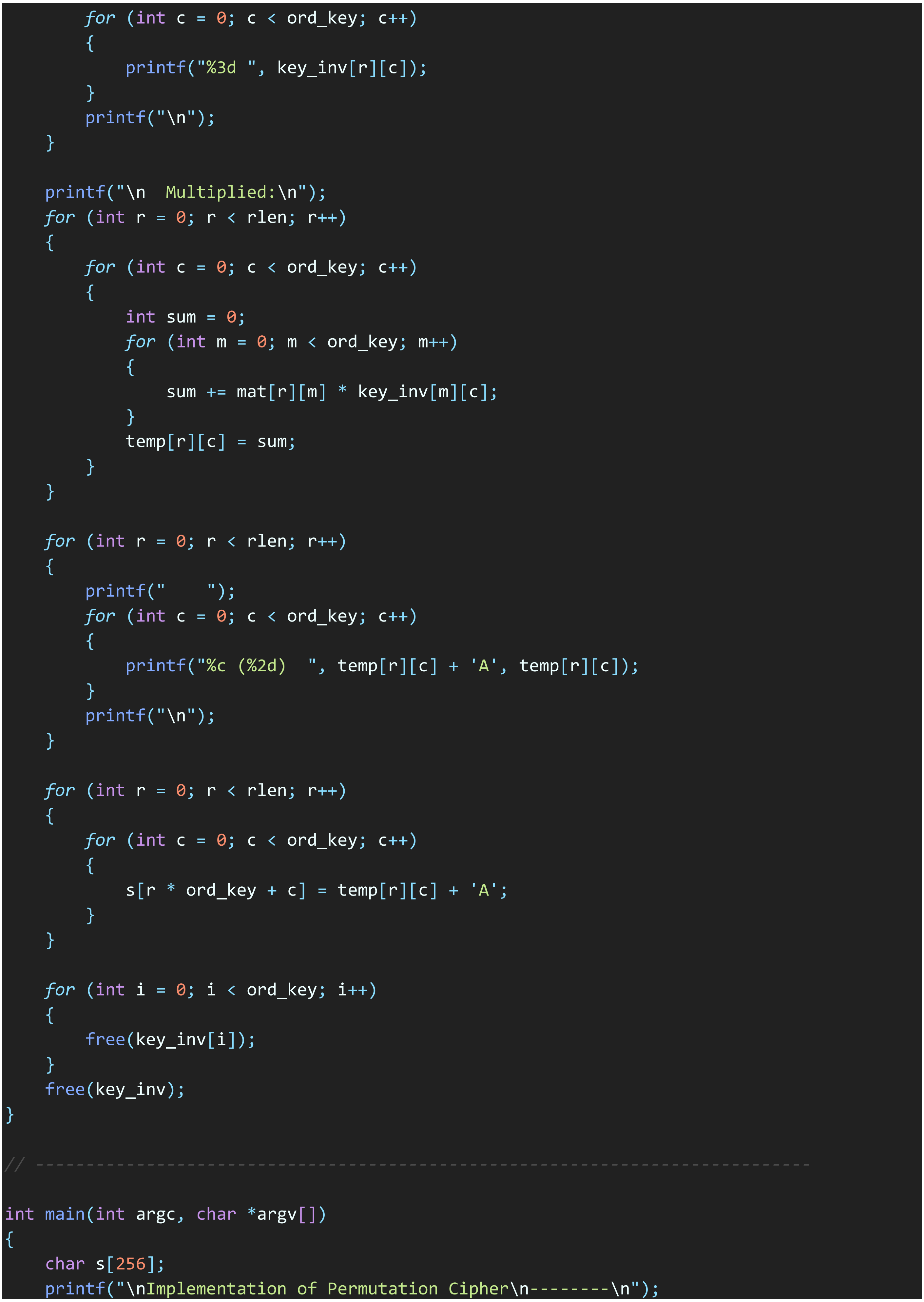
Test the operation of your encryption and decryption programs using the above π and its corresponding π. Hence decrypt the following cipher text, which was encrypted using the above π:

TGEEMNELNNTDROEOAAHDOETCSHAEIRLM

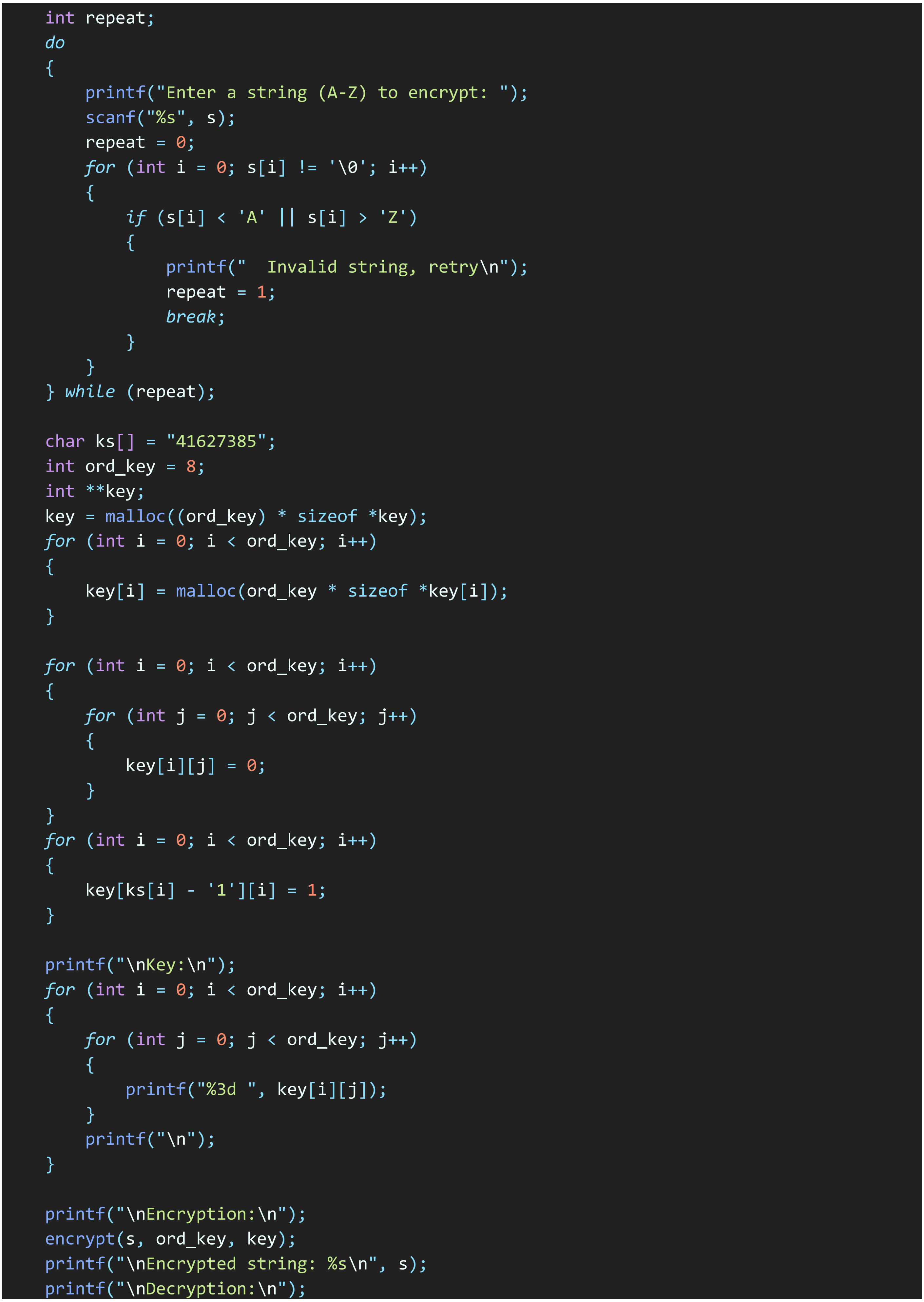
HINT: The key of a transposition cipher may be represented as a matrix of zeros and ones.

Source: `3-cipher-permutation.c`

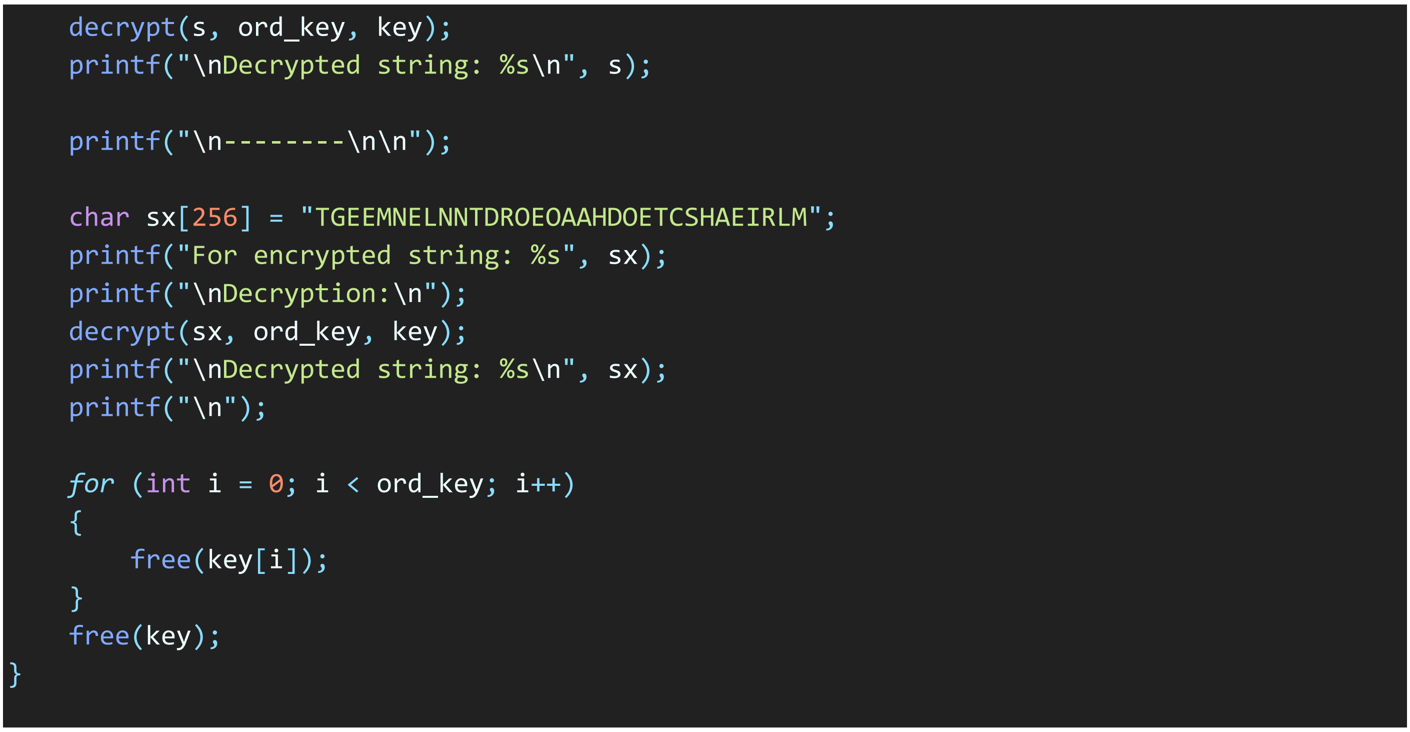
```c
#include <stdio.h>  // printf, scanf
#include <stdlib.h> // abs, srand, rand
#include <string.h> // strlen
#include <time.h>   // time

#include "../utils.h"
#define MOD 26

/*
 * Cipher permutation 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)
{
    int len = strlen(s);
    int rlen = len / ord_key + (len % ord_key == 0 ? 0 : 1);

    int mat[rlen][ord_key];
    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < ord_key; c++)
        {
            int pos = r * ord_key + c;
            mat[r][c] = pos < len ? s[pos] - 'A' : 25;
        }
    }

    printf("  Initial:\n");
    for (int r = 0; r < rlen; r++)
    {
        printf("    ");
        for (int c = 0; c < ord_key; c++)
        {
            printf("%c (%2d)  ", mat[r][c] + 'A', mat[r][c]);
        }
        printf("\n");
    }

    int temp[rlen][ord_key];

    printf("\n  Multiplied:\n");
    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < ord_key; c++)
        {
            int sum = 0;
            for (int m = 0; m < ord_key; m++)
            {
                sum += mat[r][m] * key[m][c];
            }
            temp[r][c] = sum;
        }
    }

    for (int r = 0; r < rlen; r++)
    {
        printf("    ");
        for (int c = 0; c < ord_key; c++)
        {
            printf("%c (%2d)  ", temp[r][c] + 'A', temp[r][c]);
        }
        printf("\n");
    }

    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < ord_key; c++)
        {
            s[r * ord_key + c] = temp[r][c] + 'A';
        }
    }
    s[ord_key * rlen] = '\0';
}

/*
 * Cipher permutation 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)
{
    int len = strlen(s);
    int rlen = len / ord_key + (len % ord_key == 0 ? 0 : 1);

    int mat[rlen][ord_key];
    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < ord_key; c++)
        {
            mat[r][c] = s[r * ord_key + c] - 'A';
        }
    }

    int temp[rlen][ord_key];

    char k[ord_key];
    for (int c = 0; c < ord_key; c++)
    {
        for (int r = 0; r < ord_key; r++)
        {
            if (key[r][c] == 1)
            {
                k[c] = r + '1';
                break;
            }
        }
    }
    char ki[ord_key];
    for (int i = 0; i < ord_key; i++)
    {
        ki[k[i] - '1'] = i + '1';
    }

    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);

    for (int c = 0; c < ord_key; c++)
    {
        for (int r = 0; r < ord_key; r++)
        {
            key_inv[r][c] = 0;
        }
    }
    for (int i = 0; i < ord_key; i++)
    {
        key_inv[ki[i] - '1'][i] = 1;
    }

    printf("  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");
    }

    printf("\n  Multiplied:\n");
    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < ord_key; c++)
        {
            int sum = 0;
            for (int m = 0; m < ord_key; m++)
            {
                sum += mat[r][m] * key_inv[m][c];
            }
            temp[r][c] = sum;
        }
    }

    for (int r = 0; r < rlen; r++)
    {
        printf("    ");
        for (int c = 0; c < ord_key; c++)
        {
            printf("%c (%2d)  ", temp[r][c] + 'A', temp[r][c]);
        }
        printf("\n");
    }

    for (int r = 0; r < rlen; r++)
    {
        for (int c = 0; c < ord_key; c++)
        {
            s[r * ord_key + c] = temp[r][c] + '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 Permutation Cipher\n--------\n");
```

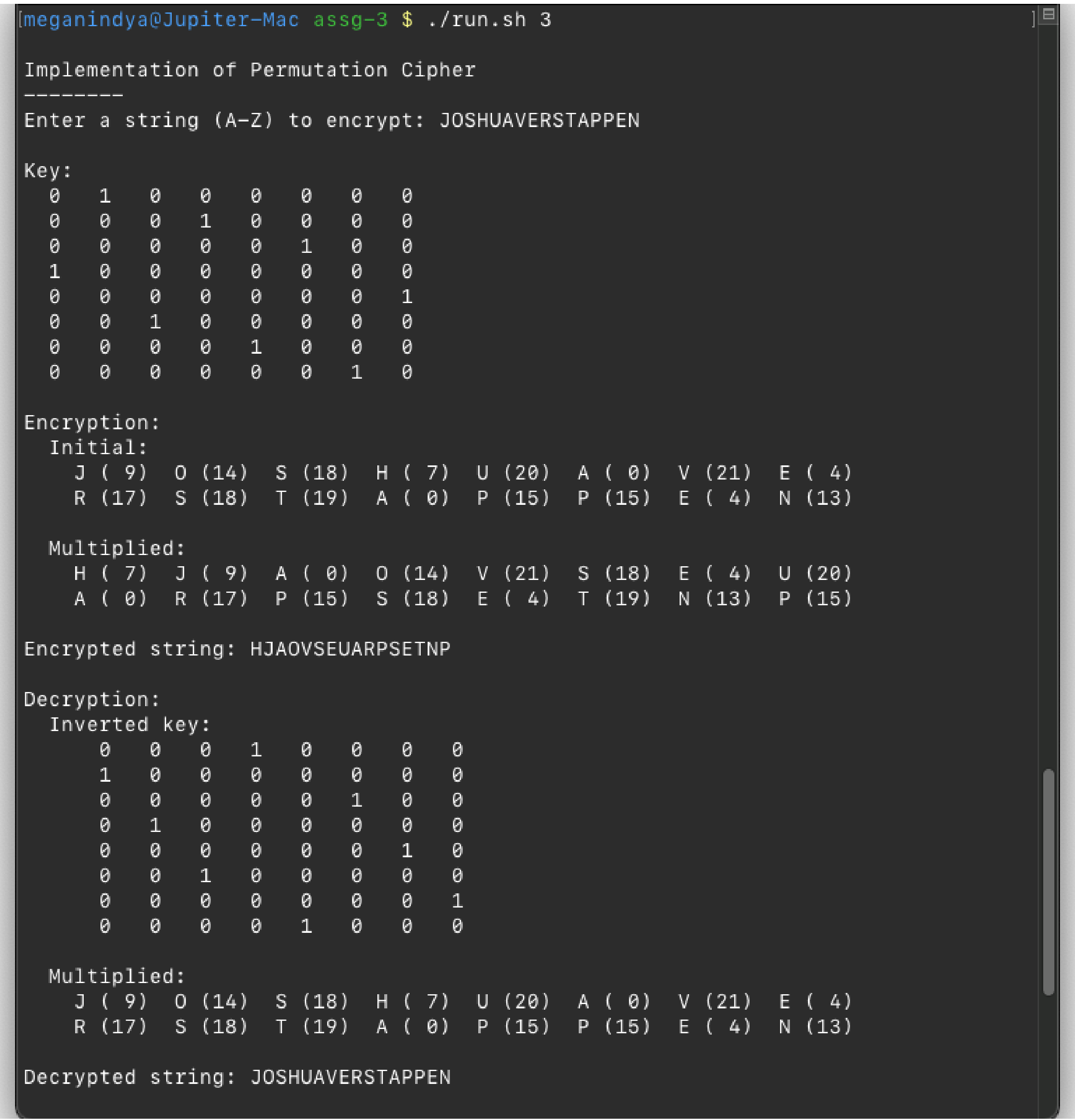


```
    int repeat;
    do
    {
        printf("Enter a string (A-Z) to encrypt: ");
        scanf("%s", s);
        repeat = 0;
        for (int i = 0; s[i] != '\0'; i++)
        {
            if (s[i] < 'A' || s[i] > 'Z')
            {
                printf("  Invalid string, retry\n");
        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

    char ks[] = "41627385";
    int ord_key = 8;
    int **key;
    key = malloc((ord_key) * sizeof *key);
    for (int i = 0; i < ord_key; i++)
    {
        key[i] = malloc(ord_key * sizeof *key[i]);
    }

    for (int i = 0; i < ord_key; i++)
    {
        for (int j = 0; j < ord_key; j++)
        {
            key[i][j] = 0;
        }
    }
    for (int i = 0; i < ord_key; i++)
    {
        key[ks[i] - '1'][i] = 1;
    }

    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");
```

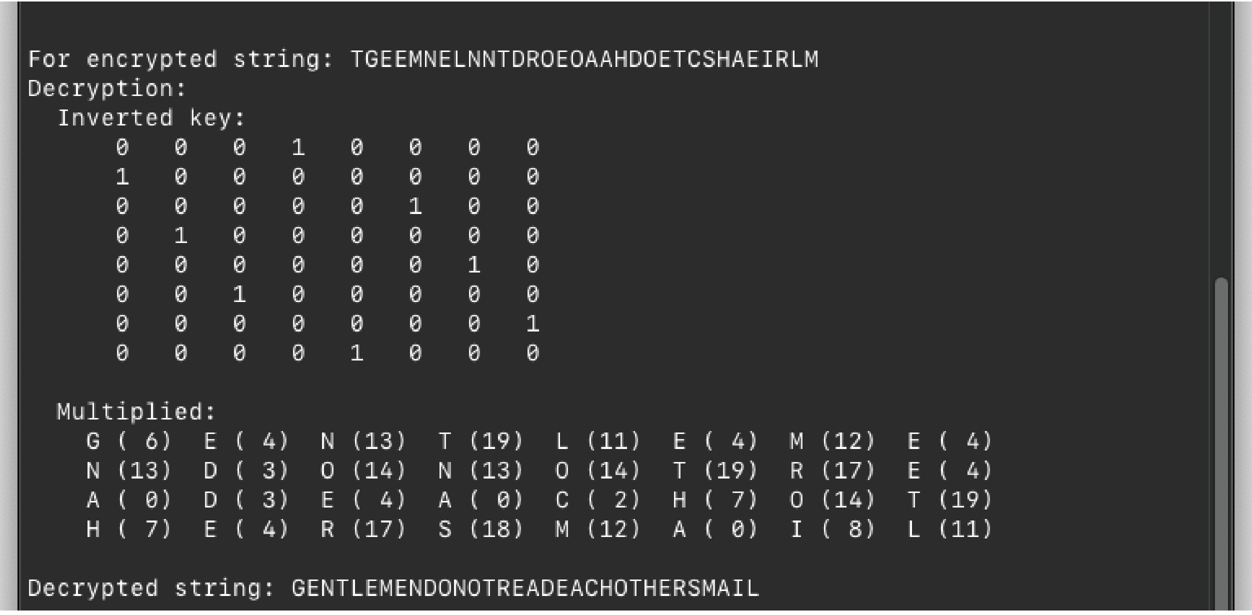


```
    decrypt(s, ord_key, key);
    printf("\nDecrypted string: %s\n", s);

    printf("\n--------\n\n");

    char sx[256] = "TGEEMNELNNTDROEOAAHDOETCSHAEIRLM";
    printf("For encrypted string: %s", sx);
    printf("\nDecryption:\n");
    decrypt(sx, ord_key, key);
    printf("\nDecrypted string: %s\n", sx);
    printf("\n");

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

Sample run



```
meganindya@Jupiter-Mac assg-3 $ ./run.sh 3

Implementation of Permutation Cipher
--------
Enter a string (A-Z) to encrypt: JOSHUAVERSTAPPEN

Key:
  0   1   0   0   0   0   0   0
  0   0   0   1   0   0   0   0
  0   0   0   0   0   1   0   0
  1   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   1
  0   0   1   0   0   0   0   0
  0   0   0   0   1   0   0   0
  0   0   0   0   0   0   1   0

Encryption:
  Initial:
    J ( 9)  O (14)  S (18)  H ( 7)  U (20)  A ( 0)  V (21)  E ( 4)
    R (17)  S (18)  T (19)  A ( 0)  P (15)  P (15)  E ( 4)  N (13)

  Multiplied:
    H ( 7)  J ( 9)  A ( 0)  O (14)  V (21)  S (18)  E ( 4)  U (20)
    A ( 0)  R (17)  P (15)  S (18)  E ( 4)  T (19)  N (13)  P (15)

Encrypted string: HJAOVSEUARPSETNP

Decryption:
  Inverted key:
      0   0   0   1   0   0   0   0
      1   0   0   0   0   0   0   0
      0   0   0   0   0   1   0   0
      0   1   0   0   0   0   0   0
      0   0   0   0   0   0   1   0
      0   0   1   0   0   0   0   0
      0   0   0   0   0   0   0   1
      0   0   0   0   1   0   0   0

  Multiplied:
    J ( 9)  O (14)  S (18)  H ( 7)  U (20)  A ( 0)  V (21)  E ( 4)
    R (17)  S (18)  T (19)  A ( 0)  P (15)  P (15)  E ( 4)  N (13)

Decrypted string: JOSHUAVERSTAPPEN
```



```
For encrypted string: TGEEMNELNNTDROEOAAHDOETCSHAEIRLM
Decryption:
  Inverted key:
      0   0   0   1   0   0   0   0
      1   0   0   0   0   0   0   0
      0   0   0   0   0   1   0   0
      0   1   0   0   0   0   0   0
      0   0   0   0   0   0   1   0
      0   0   1   0   0   0   0   0
      0   0   0   0   0   0   0   1
      0   0   0   0   1   0   0   0

  Multiplied:
    G ( 6)  E ( 4)  N (13)  T (19)  L (11)  E ( 4)  M (12)  E ( 4)
    N (13)  D ( 3)  O (14)  N (13)  O (14)  T (19)  R (17)  E ( 4)
    A ( 0)  D ( 3)  E ( 4)  A ( 0)  C ( 2)  H ( 7)  O (14)  T (19)
    H ( 7)  E ( 4)  R (17)  S (18)  M (12)  A ( 0)  I ( 8)  L (11)

Decrypted string: GENTLEMENDONOTREADEACHOTHERSMAIL
```

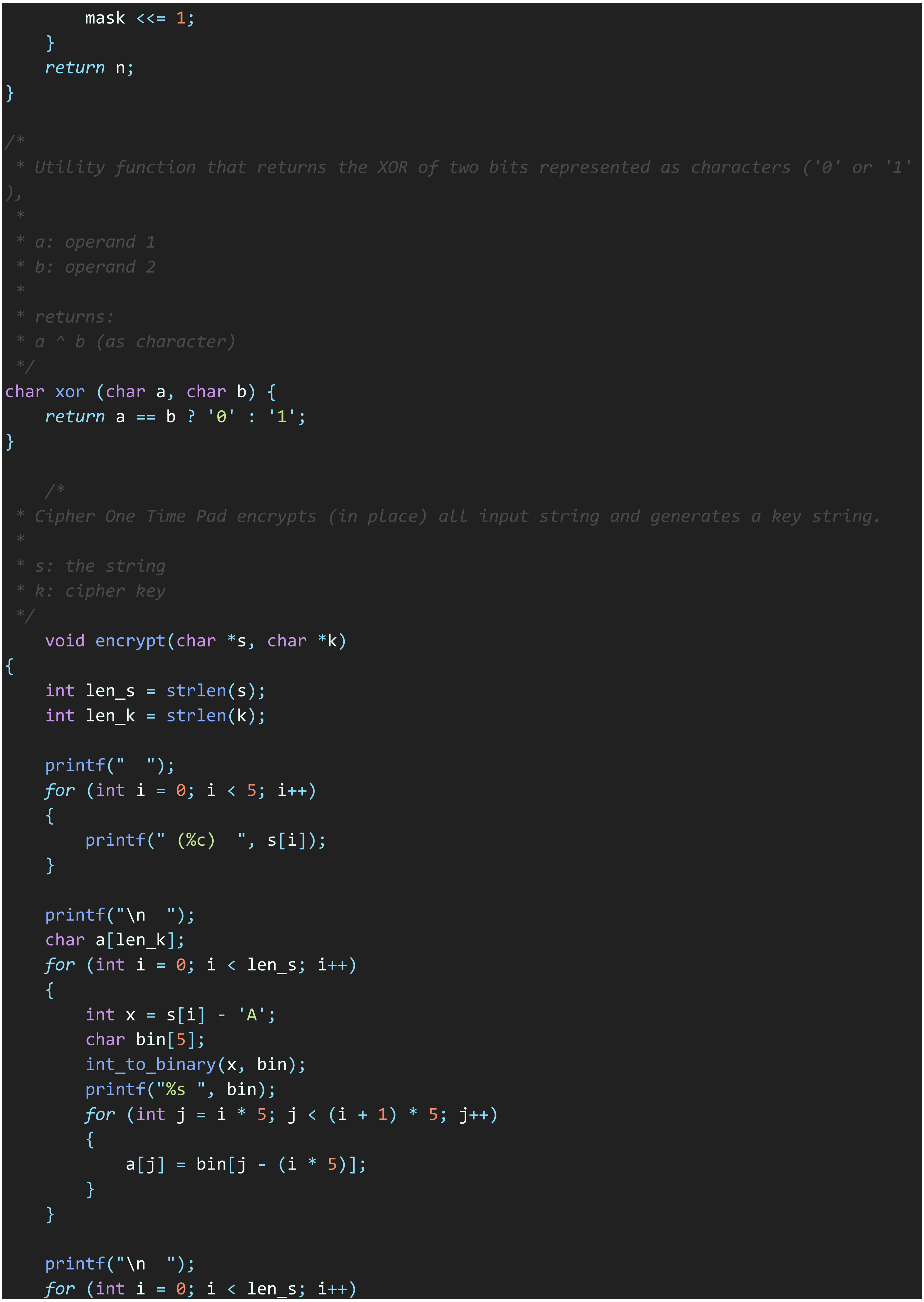
4. A One-Time Pad (OTP) is a stream cipher which uses True Random Number Generator (TRNG) to generate its key-stream, hence the name OTP. It uses the XOR-operation as both encryption and decryption functions.

   a) Implement an OTP.

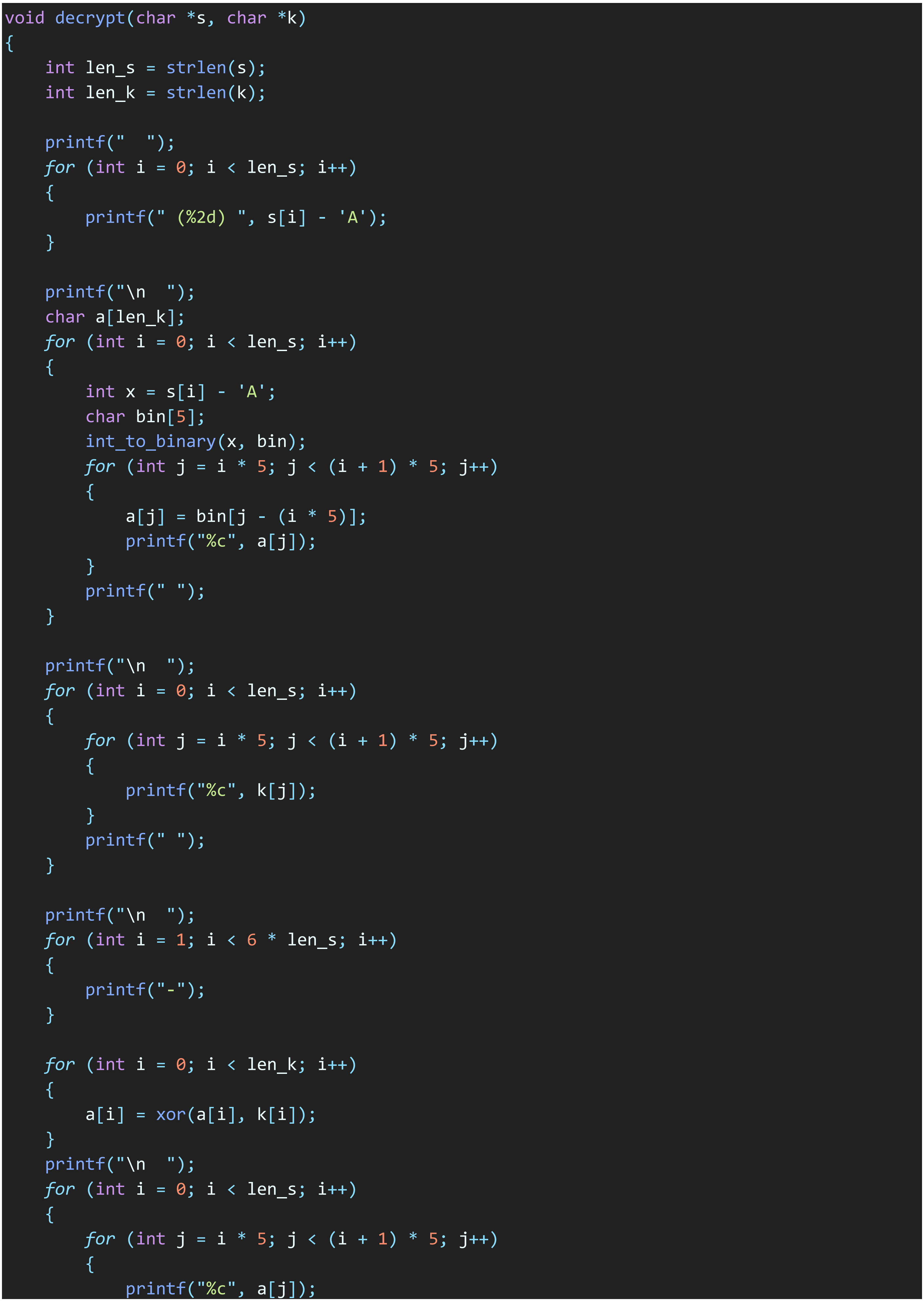
   (You may use any Pseudo-Random Number Generator (PRNG) to generate the key-stream here considering that following a physical process is infeasible for this laboratory.)

Source: `4-a-cipher-one-time-pad.c`

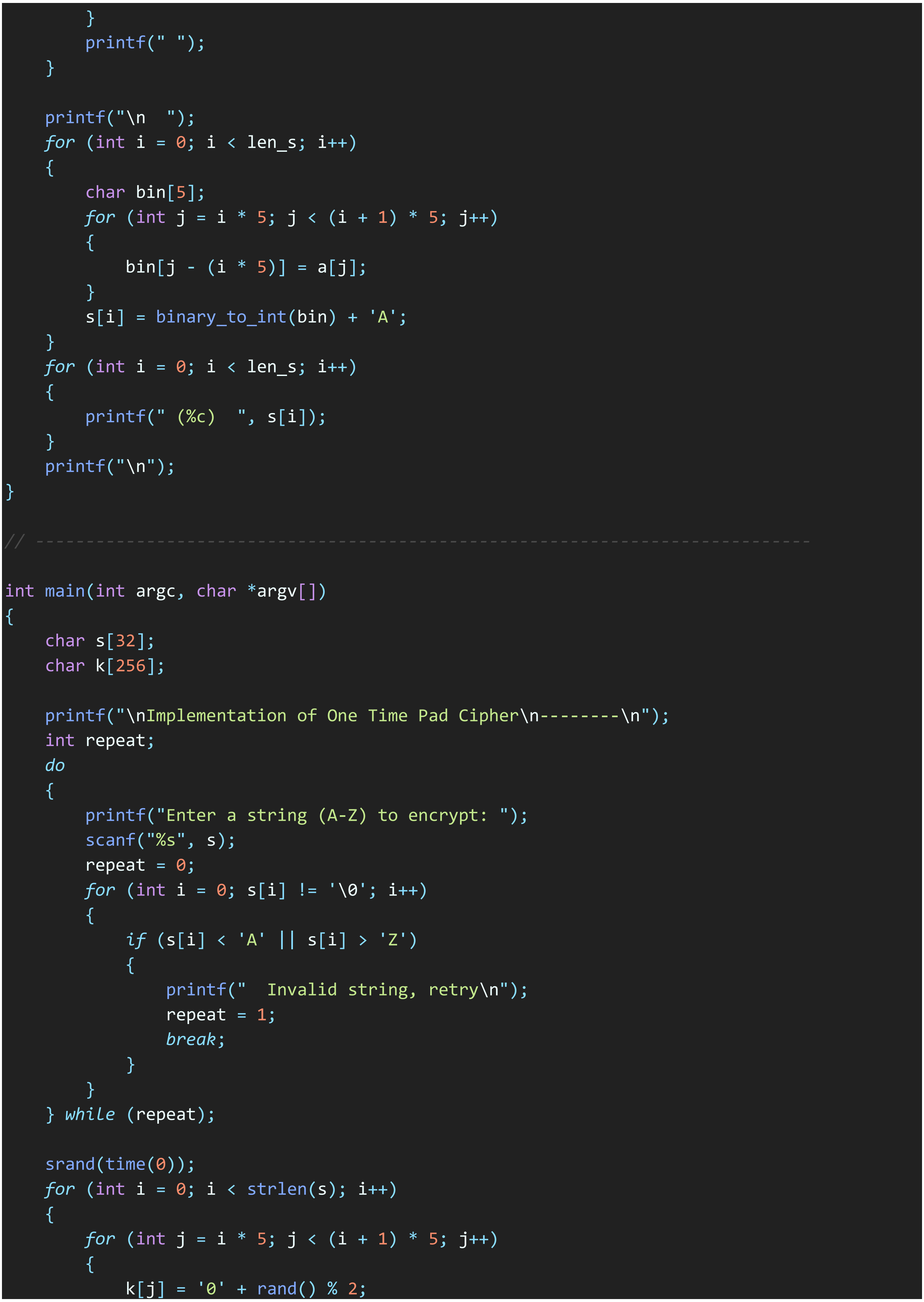
```c
#include <stdio.h>  // printf, scanf
#include <stdlib.h> // abs
#include <string.h> // strlen
#include <time.h>   // time

#include "../utils.h"
#define MOD 26

/*
 * Utility function that converts an integer (base 10) to binary (base 2) string.
 * There is a condition that the integers represent alphabets and are therefore confined i
n the
 * range [0, 25], which can be covered in 5 bits.
 *
 * n: integer number
 * s: character array to fill (binary) bits in (array length assumed to be 5)
 */
void int_to_binary(int n, char *s)
{
    int mask = 1;
    for (int i = 0; i < 5; i++)
    {
        s[4 - i] = (n & mask) == 0 ? '0' : '1';
        mask <<= 1;
    }
}

/*
 * Utility function that converts a binary (base 2) string to integer (base 10).
 * There is a condition that the integers represent alphabets and are therefore confined i
n the
 * range [0, 25], which can be covered in 5 bits.
 *
 * s: character array of bits representing the binary (array length assumed to be 5)
 *
 * returns:
 * integer (base 10) equivalent
 */
int binary_to_int(char *s)
{
    int n = 0, mask = 1;
    for (int i = 0; i < 5; i++)
    {
        n += (s[4 - i] - '0') * mask;
```



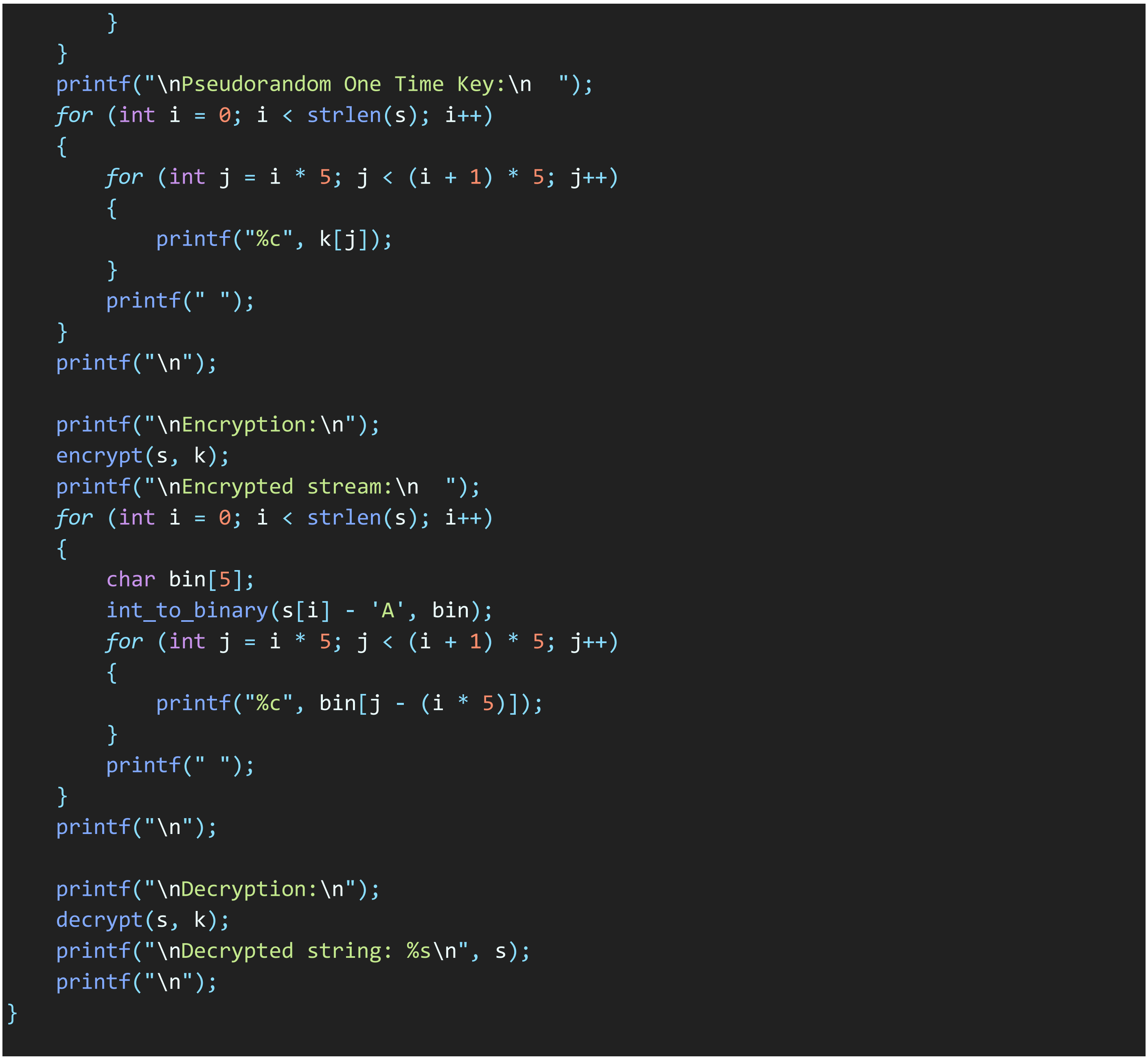
```
        mask <<= 1;
    }
    return n;
}

/*
 * Utility function that returns the XOR of two bits represented as characters ('0' or '1'
),
 *
 * a: operand 1
 * b: operand 2
 *
 * returns:
 * a ^ b (as character)
 */
char xor (char a, char b) {
    return a == b ? '0' : '1';
}

    /*
 * Cipher One Time Pad encrypts (in place) all input string and generates a key string.
 *
 * s: the string
 * k: cipher key
 */
    void encrypt(char *s, char *k)
{
    int len_s = strlen(s);
    int len_k = strlen(k);

    printf("  ");
    for (int i = 0; i < 5; i++)
    {
        printf(" (%c)  ", s[i]);
    }

    printf("\n  ");
    char a[len_k];
    for (int i = 0; i < len_s; i++)
    {
        int x = s[i] - 'A';
        char bin[5];
        int_to_binary(x, bin);
        printf("%s ", bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            a[j] = bin[j - (i * 5)];
        }
    }

    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", k[j]);
        }
        printf(" ");
    }

    printf("\n  ");
    for (int i = 1; i < 6 * len_s; i++)
    {
        printf("-");
    }

    for (int i = 0; i < len_k; i++)
    {
        a[i] = xor(a[i], k[i]);
    }
    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", a[j]);
        }
        printf(" ");
    }

    for (int i = 0; i < len_s; i++)
    {
        char bin[5];
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            bin[j - (i * 5)] = a[j];
        }
        s[i] = binary_to_int(bin) + 'A';
    }

    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        printf(" (%2d) ", s[i] - 'A');
    }
    printf("\n");
}

/*
 * Cipher One Time Pad decrypts (in place) all characters of a string.
 *
 * s: the string
 * k: cipher key
 */
```



```c
void decrypt(char *s, char *k)
{
    int len_s = strlen(s);
    int len_k = strlen(k);

    printf("  ");
    for (int i = 0; i < len_s; i++)
    {
        printf(" (%2d) ", s[i] - 'A');
    }

    printf("\n  ");
    char a[len_k];
    for (int i = 0; i < len_s; i++)
    {
        int x = s[i] - 'A';
        char bin[5];
        int_to_binary(x, bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            a[j] = bin[j - (i * 5)];
            printf("%c", a[j]);
        }
        printf(" ");
    }

    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", k[j]);
        }
        printf(" ");
    }

    printf("\n  ");
    for (int i = 1; i < 6 * len_s; i++)
    {
        printf("-");
    }

    for (int i = 0; i < len_k; i++)
    {
        a[i] = xor(a[i], k[i]);
    }
    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", a[j]);
```



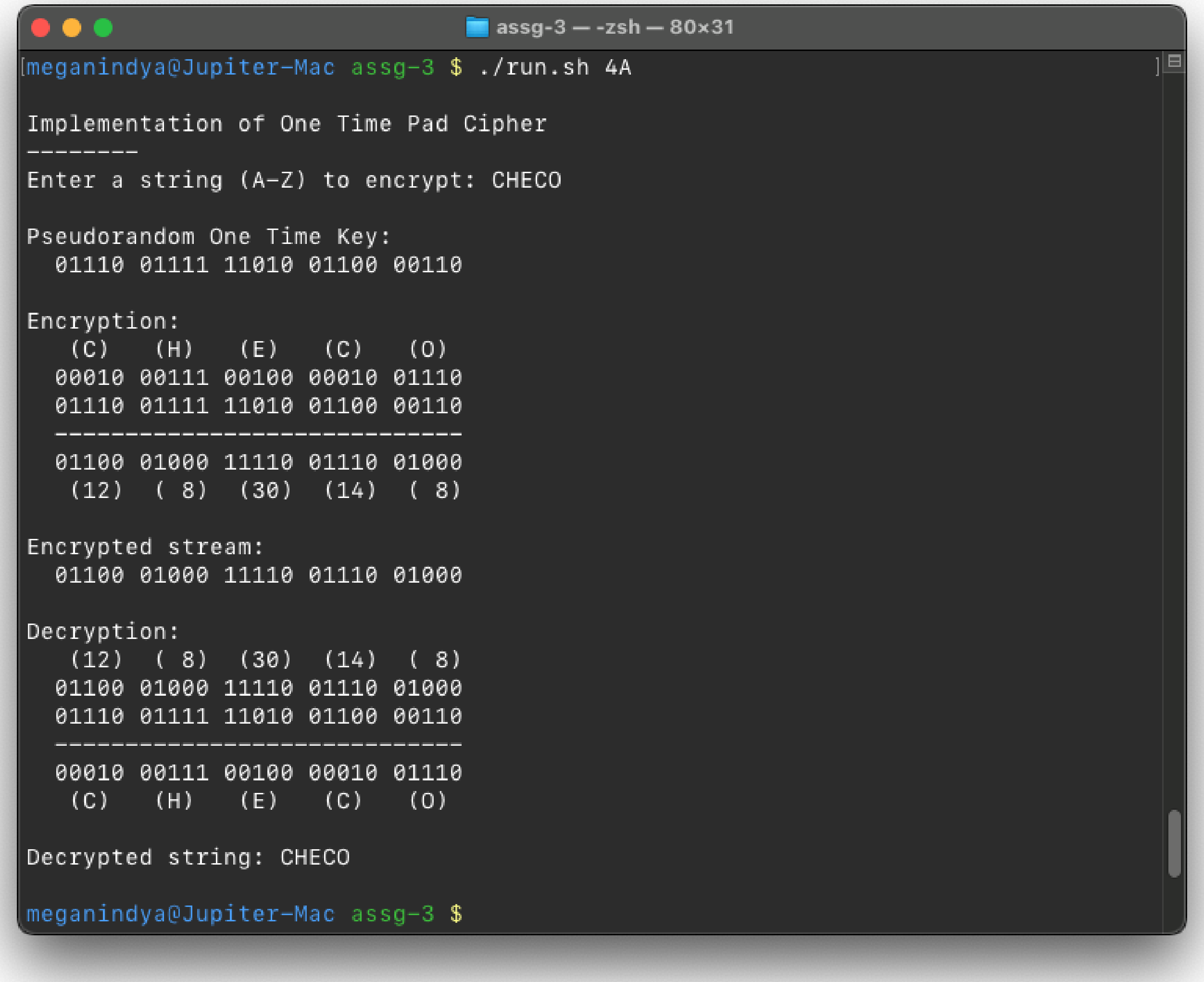
```c
        }
        printf(" ");
    }

    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        char bin[5];
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            bin[j - (i * 5)] = a[j];
        }
        s[i] = binary_to_int(bin) + 'A';
    }
    for (int i = 0; i < len_s; i++)
    {
        printf(" (%c)  ", s[i]);
    }
    printf("\n");
}

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

int main(int argc, char *argv[])
{
    char s[32];
    char k[256];

    printf("\nImplementation of One Time Pad 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);

    srand(time(0));
    for (int i = 0; i < strlen(s); i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            k[j] = '0' + rand() % 2;
```



```c
        }
    }
    printf("\nPseudorandom One Time Key:\n  ");
    for (int i = 0; i < strlen(s); i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", k[j]);
        }
        printf(" ");
    }
    printf("\n");

    printf("\nEncryption:\n");
    encrypt(s, k);
    printf("\nEncrypted stream:\n  ");
    for (int i = 0; i < strlen(s); i++)
    {
        char bin[5];
        int_to_binary(s[i] - 'A', bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", bin[j - (i * 5)]);
        }
        printf(" ");
    }
    printf("\n");

    printf("\nDecryption:\n");
    decrypt(s, k);
    printf("\nDecrypted string: %s\n", s);
    printf("\n");
}
```

[P.T.O.]

Sample run



```
meganindya@Jupiter-Mac assg-3 $ ./run.sh 4A

Implementation of One Time Pad Cipher
--------
Enter a string (A-Z) to encrypt: CHECO

Pseudorandom One Time Key:
  01110 01111 11010 01100 00110

Encryption:
   (C)   (H)   (E)   (C)   (O)
  00010 00111 00100 00010 01110
  01110 01111 11010 01100 00110
  -----------------------------
  01100 01000 11110 01110 01000
   (12)  ( 8)  (30)  (14)  ( 8)

Encrypted stream:
  01100 01000 11110 01110 01000

Decryption:
   (12)  ( 8)  (30)  (14)  ( 8)
  01100 01000 11110 01110 01000
  01110 01111 11010 01100 00110
  -----------------------------
  00010 00111 00100 00010 01110
   (C)   (H)   (E)   (C)   (O)

Decrypted string: CHECO

meganindya@Jupiter-Mac assg-3 $
```

b) Assuming the PRNG looks like:

$S_0 = seed$

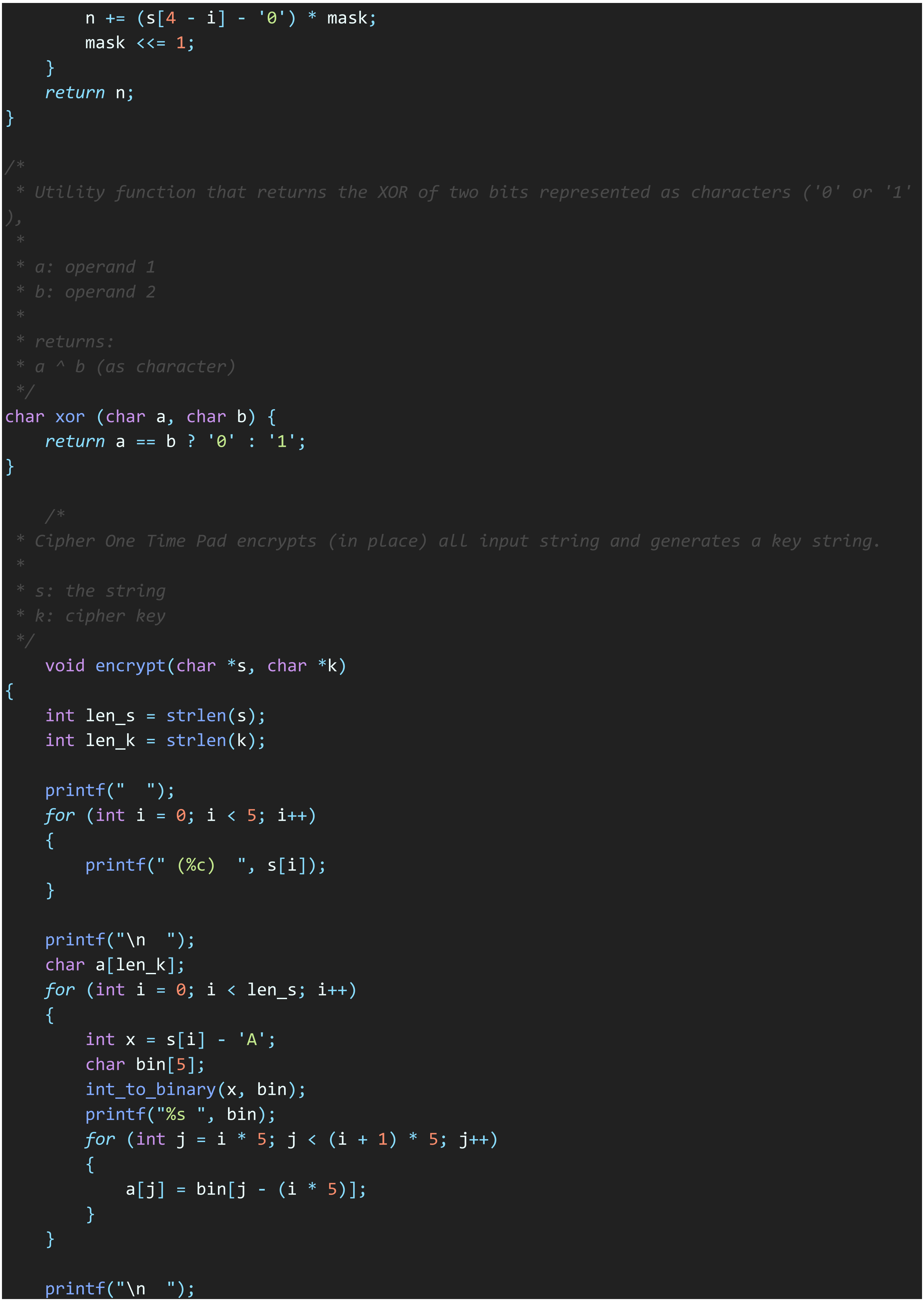
$S_{i+1} \equiv AS_i + B \bmod m, i = 0,1, . . .$

where m =26 is public, the secrets are A, B and the seed, where all A, B, S_i belong to Z_{26}, and an outsider is provided with the knowledge of only first 15 bits of plaintext, implement a way for (known-plaintext) cryptanalysis of the OTP.

[Hint: Note that $2_4 < 2_6 < 2_5$]

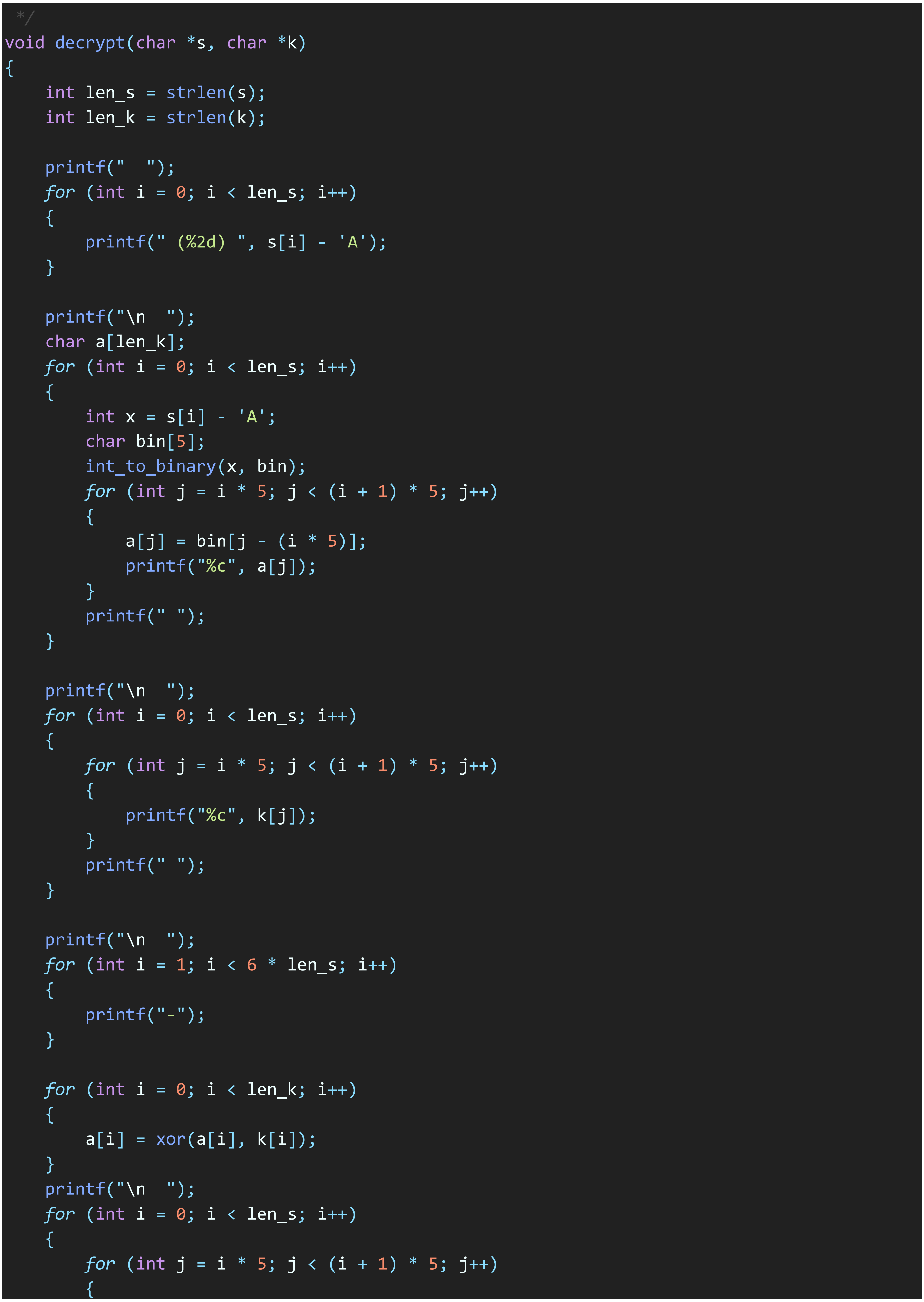
Source: `4-b-cipher-one-time-pad-cryptanalysis.c`

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

#include "../utils.h"
#define MOD 26

/*
 * Utility function that converts an integer (base 10) to binary (base 2) string.
 * There is a condition that the integers represent alphabets and are therefore confined i
n the
 * range [0, 25], which can be covered in 5 bits.
 *
 * n: integer number
 * s: character array to fill (binary) bits in (array length assumed to be 5)
 */
void int_to_binary(int n, char *s)
{
    int mask = 1;
    for (int i = 0; i < 5; i++)
    {
        s[4 - i] = (n & mask) == 0 ? '0' : '1';
        mask <<= 1;
    }
}

/*
 * Utility function that converts a binary (base 2) string to integer (base 10).
 * There is a condition that the integers represent alphabets and are therefore confined i
n the
 * range [0, 25], which can be covered in 5 bits.
 *
 * s: character array of bits representing the binary (array length assumed to be 5)
 *
 * returns:
 * integer (base 10) equivalent
 */
int binary_to_int(char *s)
{
    int n = 0, mask = 1;
    for (int i = 0; i < 5; i++)
    {
```

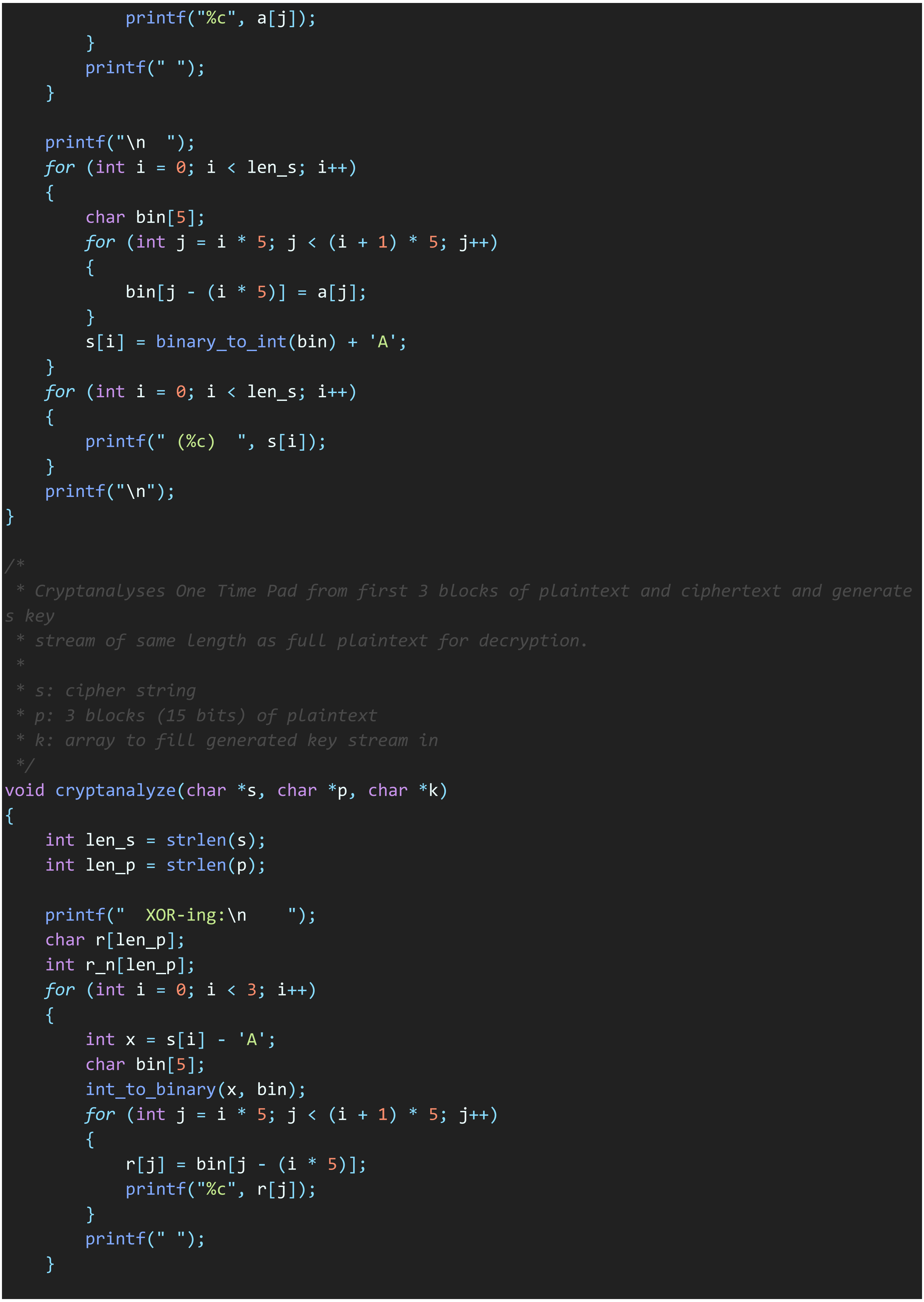


```
        n += (s[4 - i] - '0') * mask;
        mask <<= 1;
    }
    return n;
}

/*
 * Utility function that returns the XOR of two bits represented as characters ('0' or '1'
),
 *
 * a: operand 1
 * b: operand 2
 *
 * returns:
 * a ^ b (as character)
 */
char xor (char a, char b) {
    return a == b ? '0' : '1';
}

    /*
 * Cipher One Time Pad encrypts (in place) all input string and generates a key string.
 *
 * s: the string
 * k: cipher key
 */
    void encrypt(char *s, char *k)
{
    int len_s = strlen(s);
    int len_k = strlen(k);

    printf("  ");
    for (int i = 0; i < 5; i++)
    {
        printf(" (%c)  ", s[i]);
    }

    printf("\n  ");
    char a[len_k];
    for (int i = 0; i < len_s; i++)
    {
        int x = s[i] - 'A';
        char bin[5];
        int_to_binary(x, bin);
        printf("%s ", bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            a[j] = bin[j - (i * 5)];
        }
    }

    printf("\n  ");
```

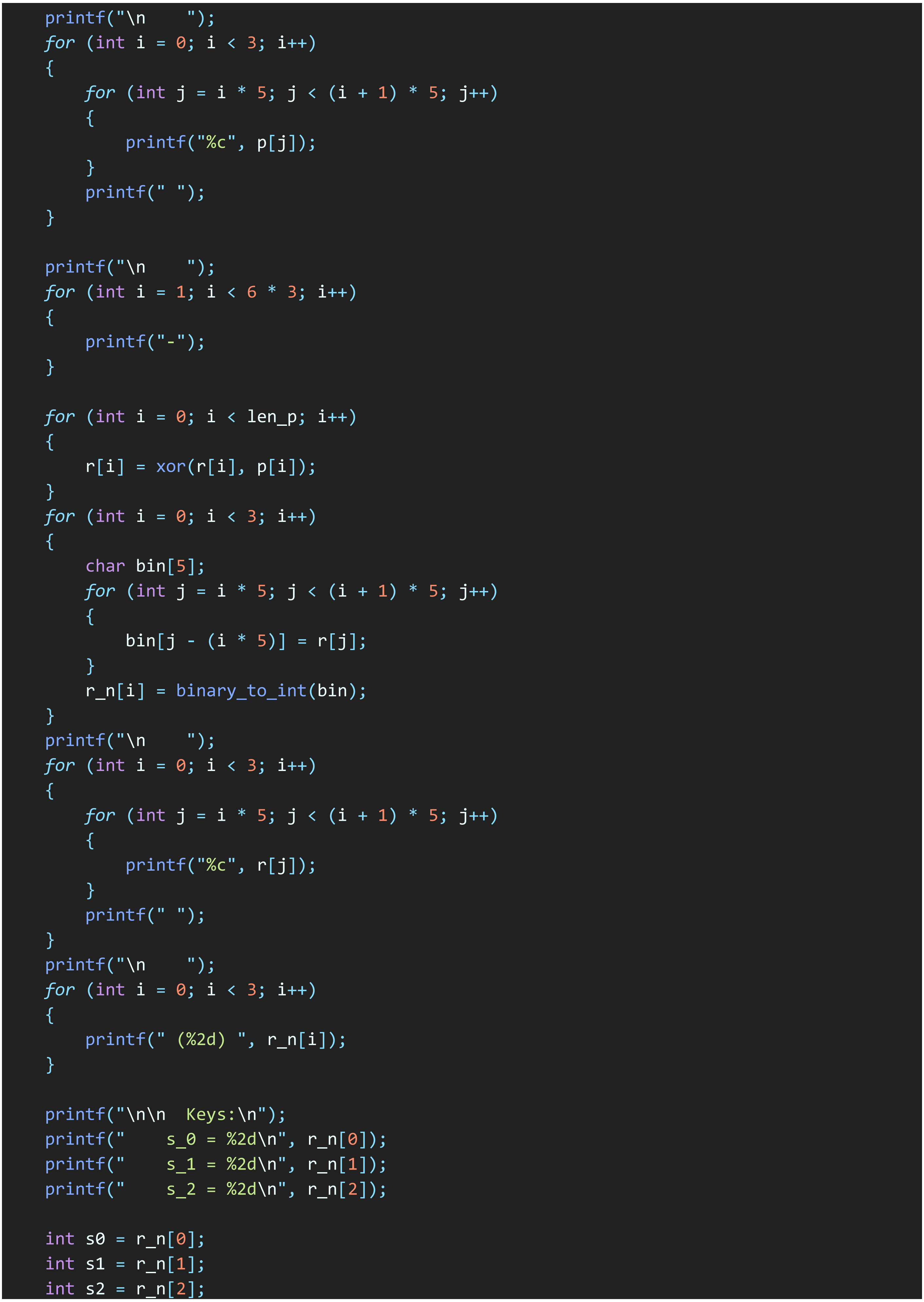
```c
    for (int i = 0; i < len_s; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", k[j]);
        }
        printf(" ");
    }

    printf("\n  ");
    for (int i = 1; i < 6 * len_s; i++)
    {
        printf("-");
    }

    for (int i = 0; i < len_k; i++)
    {
        a[i] = xor(a[i], k[i]);
    }
    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", a[j]);
        }
        printf(" ");
    }

    for (int i = 0; i < len_s; i++)
    {
        char bin[5];
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            bin[j - (i * 5)] = a[j];
        }
        s[i] = binary_to_int(bin) + 'A';
    }

    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        printf(" (%2d) ", s[i] - 'A');
    }
    printf("\n");
}

/*
 * Cipher One Time Pad decrypts (in place) all characters of a string.
 *
 * s: the string
 * k: cipher key
```



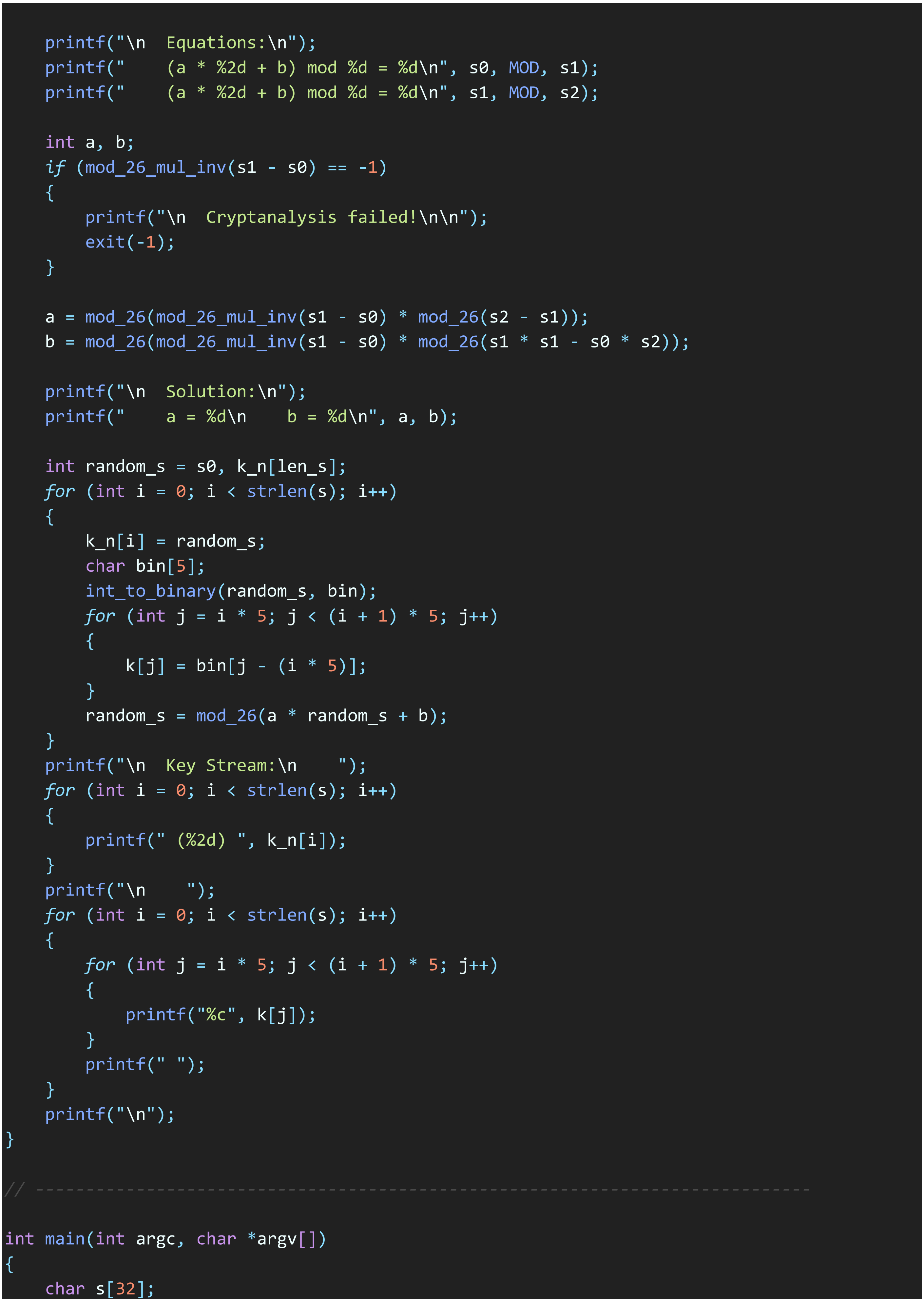
```
 */
void decrypt(char *s, char *k)
{
    int len_s = strlen(s);
    int len_k = strlen(k);

    printf("  ");
    for (int i = 0; i < len_s; i++)
    {
        printf(" (%2d) ", s[i] - 'A');
    }

    printf("\n  ");
    char a[len_k];
    for (int i = 0; i < len_s; i++)
    {
        int x = s[i] - 'A';
        char bin[5];
        int_to_binary(x, bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            a[j] = bin[j - (i * 5)];
            printf("%c", a[j]);
        }
        printf(" ");
    }

    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", k[j]);
        }
        printf(" ");
    }

    printf("\n  ");
    for (int i = 1; i < 6 * len_s; i++)
    {
        printf("-");
    }

    for (int i = 0; i < len_k; i++)
    {
        a[i] = xor(a[i], k[i]);
    }
    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
```



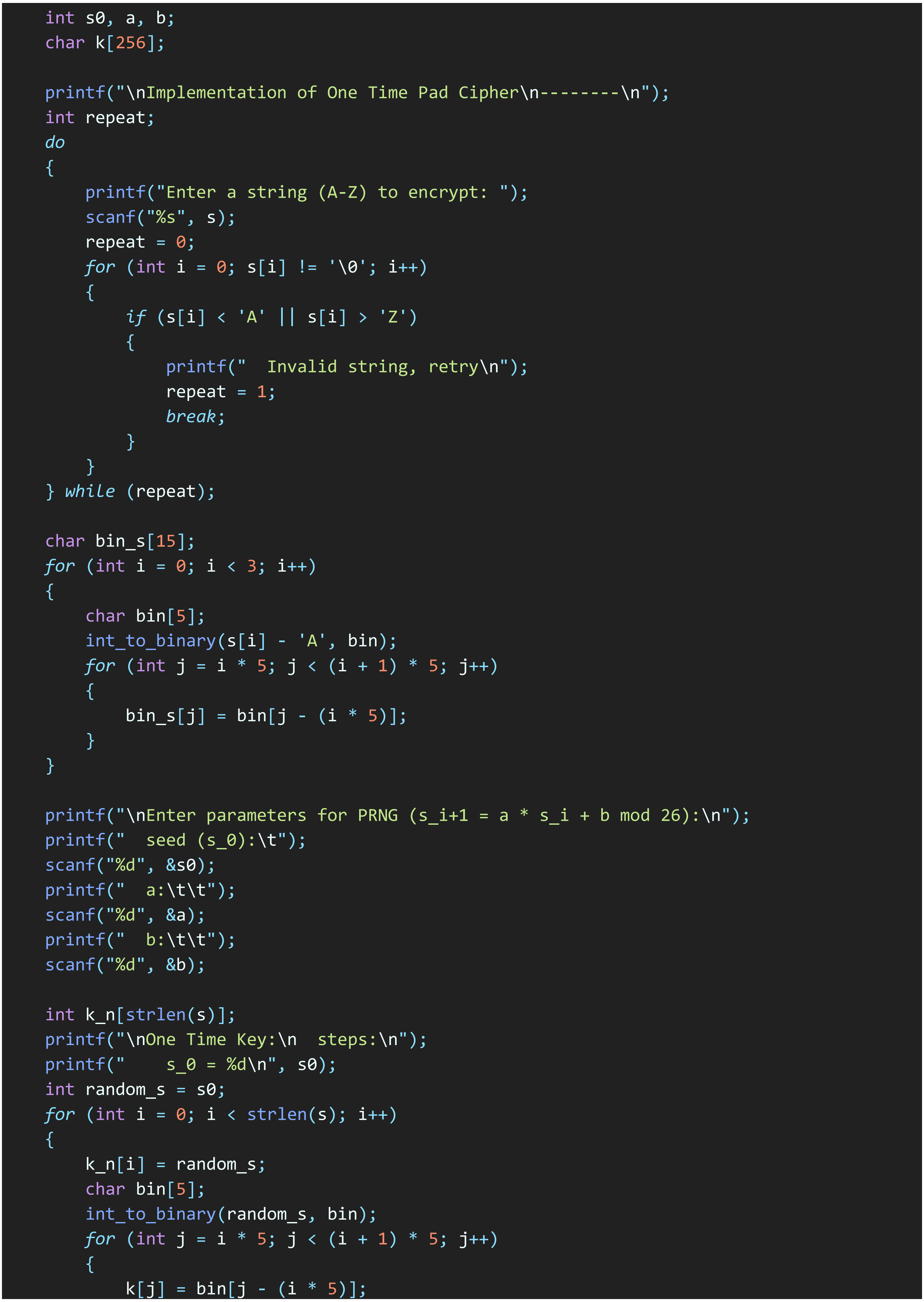
```
            printf("%c", a[j]);
        }
        printf(" ");
    }

    printf("\n  ");
    for (int i = 0; i < len_s; i++)
    {
        char bin[5];
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            bin[j - (i * 5)] = a[j];
        }
        s[i] = binary_to_int(bin) + 'A';
    }
    for (int i = 0; i < len_s; i++)
    {
        printf(" (%c)  ", s[i]);
    }
    printf("\n");
}

/*
 * Cryptanalyses One Time Pad from first 3 blocks of plaintext and ciphertext and generate
s key
 * stream of same length as full plaintext for decryption.
 *
 * s: cipher string
 * p: 3 blocks (15 bits) of plaintext
 * k: array to fill generated key stream in
 */
void cryptanalyze(char *s, char *p, char *k)
{
    int len_s = strlen(s);
    int len_p = strlen(p);

    printf("  XOR-ing:\n    ");
    char r[len_p];
    int r_n[len_p];
    for (int i = 0; i < 3; i++)
    {
        int x = s[i] - 'A';
        char bin[5];
        int_to_binary(x, bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            r[j] = bin[j - (i * 5)];
            printf("%c", r[j]);
        }
        printf(" ");
    }
```



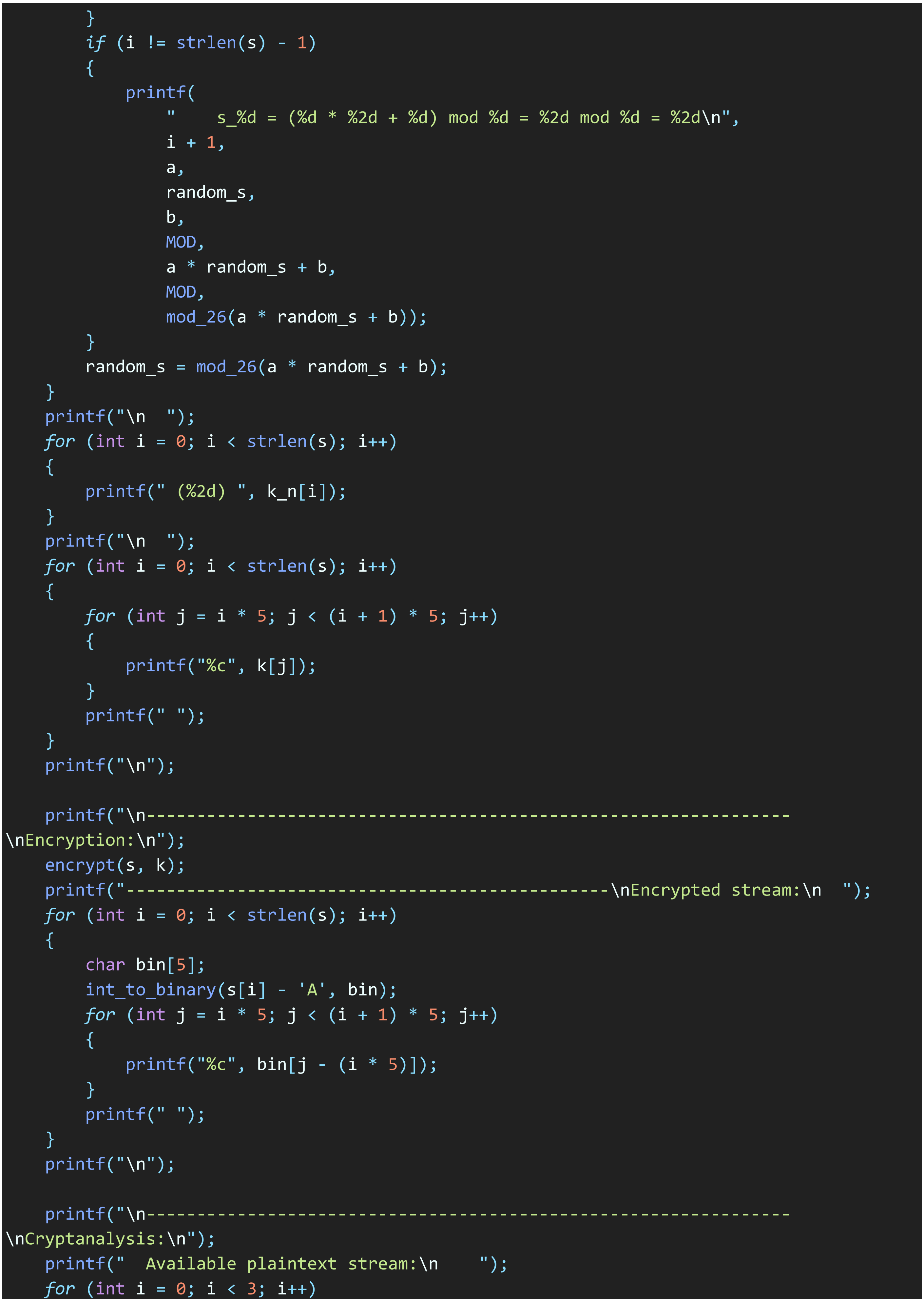
```
    printf("\n    ");
    for (int i = 0; i < 3; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", p[j]);
        }
        printf(" ");
    }

    printf("\n    ");
    for (int i = 1; i < 6 * 3; i++)
    {
        printf("-");
    }

    for (int i = 0; i < len_p; i++)
    {
        r[i] = xor(r[i], p[i]);
    }
    for (int i = 0; i < 3; i++)
    {
        char bin[5];
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            bin[j - (i * 5)] = r[j];
        }
        r_n[i] = binary_to_int(bin);
    }
    printf("\n    ");
    for (int i = 0; i < 3; i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", r[j]);
        }
        printf(" ");
    }
    printf("\n    ");
    for (int i = 0; i < 3; i++)
    {
        printf(" (%2d) ", r_n[i]);
    }

    printf("\n\n  Keys:\n");
    printf("    s_0 = %2d\n", r_n[0]);
    printf("    s_1 = %2d\n", r_n[1]);
    printf("    s_2 = %2d\n", r_n[2]);

    int s0 = r_n[0];
    int s1 = r_n[1];
    int s2 = r_n[2];
```



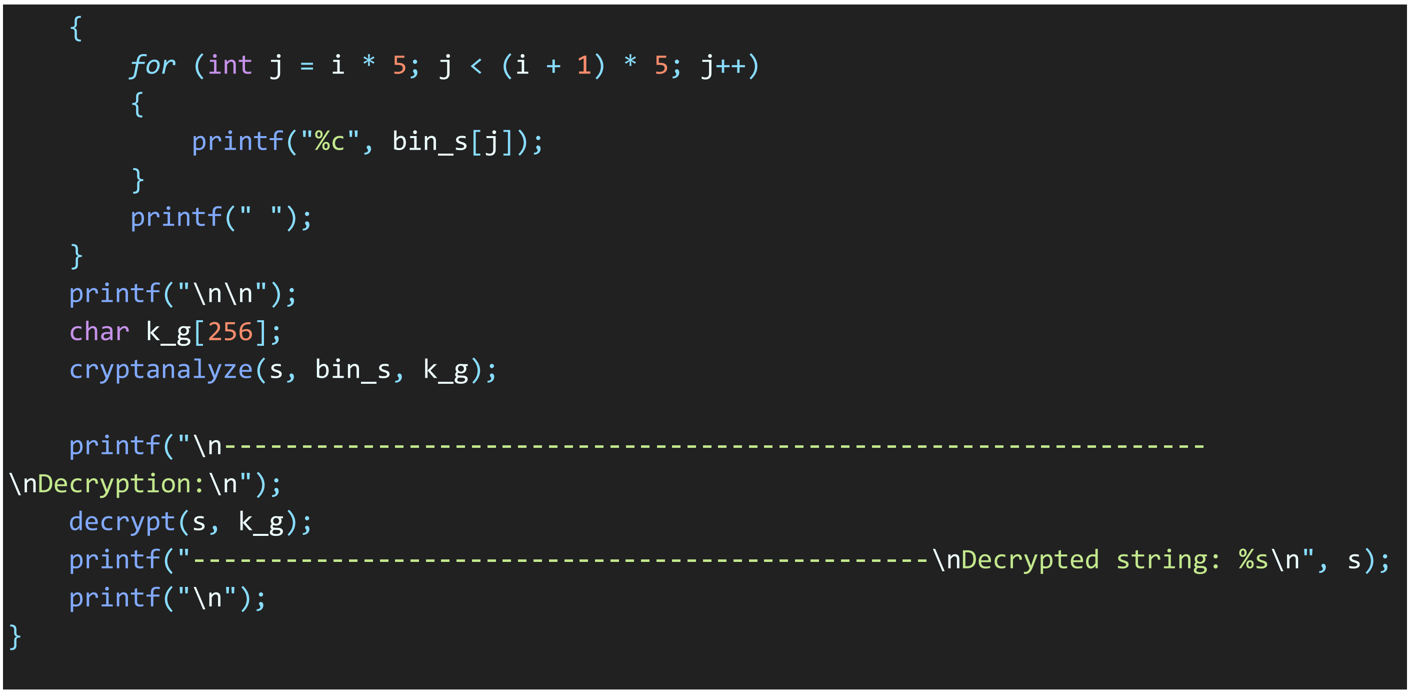
```c
    printf("\n  Equations:\n");
    printf("    (a * %2d + b) mod %d = %d\n", s0, MOD, s1);
    printf("    (a * %2d + b) mod %d = %d\n", s1, MOD, s2);

    int a, b;
    if (mod_26_mul_inv(s1 - s0) == -1)
    {
        printf("\n  Cryptanalysis failed!\n\n");
        exit(-1);
    }

    a = mod_26(mod_26_mul_inv(s1 - s0) * mod_26(s2 - s1));
    b = mod_26(mod_26_mul_inv(s1 - s0) * mod_26(s1 * s1 - s0 * s2));

    printf("\n  Solution:\n");
    printf("    a = %d\n    b = %d\n", a, b);

    int random_s = s0, k_n[len_s];
    for (int i = 0; i < strlen(s); i++)
    {
        k_n[i] = random_s;
        char bin[5];
        int_to_binary(random_s, bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            k[j] = bin[j - (i * 5)];
        }
        random_s = mod_26(a * random_s + b);
    }
    printf("\n  Key Stream:\n    ");
    for (int i = 0; i < strlen(s); i++)
    {
        printf(" (%2d) ", k_n[i]);
    }
    printf("\n    ");
    for (int i = 0; i < strlen(s); i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", k[j]);
        }
        printf(" ");
    }
    printf("\n");
}

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

int main(int argc, char *argv[])
{
    char s[32];
```



```
    int s0, a, b;
    char k[256];

    printf("\nImplementation of One Time Pad 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);

    char bin_s[15];
    for (int i = 0; i < 3; i++)
    {
        char bin[5];
        int_to_binary(s[i] - 'A', bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            bin_s[j] = bin[j - (i * 5)];
        }
    }

    printf("\nEnter parameters for PRNG (s_i+1 = a * s_i + b mod 26):\n");
    printf("  seed (s_0):\t");
    scanf("%d", &s0);
    printf("  a:\t\t");
    scanf("%d", &a);
    printf("  b:\t\t");
    scanf("%d", &b);

    int k_n[strlen(s)];
    printf("\nOne Time Key:\n  steps:\n");
    printf("    s_0 = %d\n", s0);
    int random_s = s0;
    for (int i = 0; i < strlen(s); i++)
    {
        k_n[i] = random_s;
        char bin[5];
        int_to_binary(random_s, bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            k[j] = bin[j - (i * 5)];
```

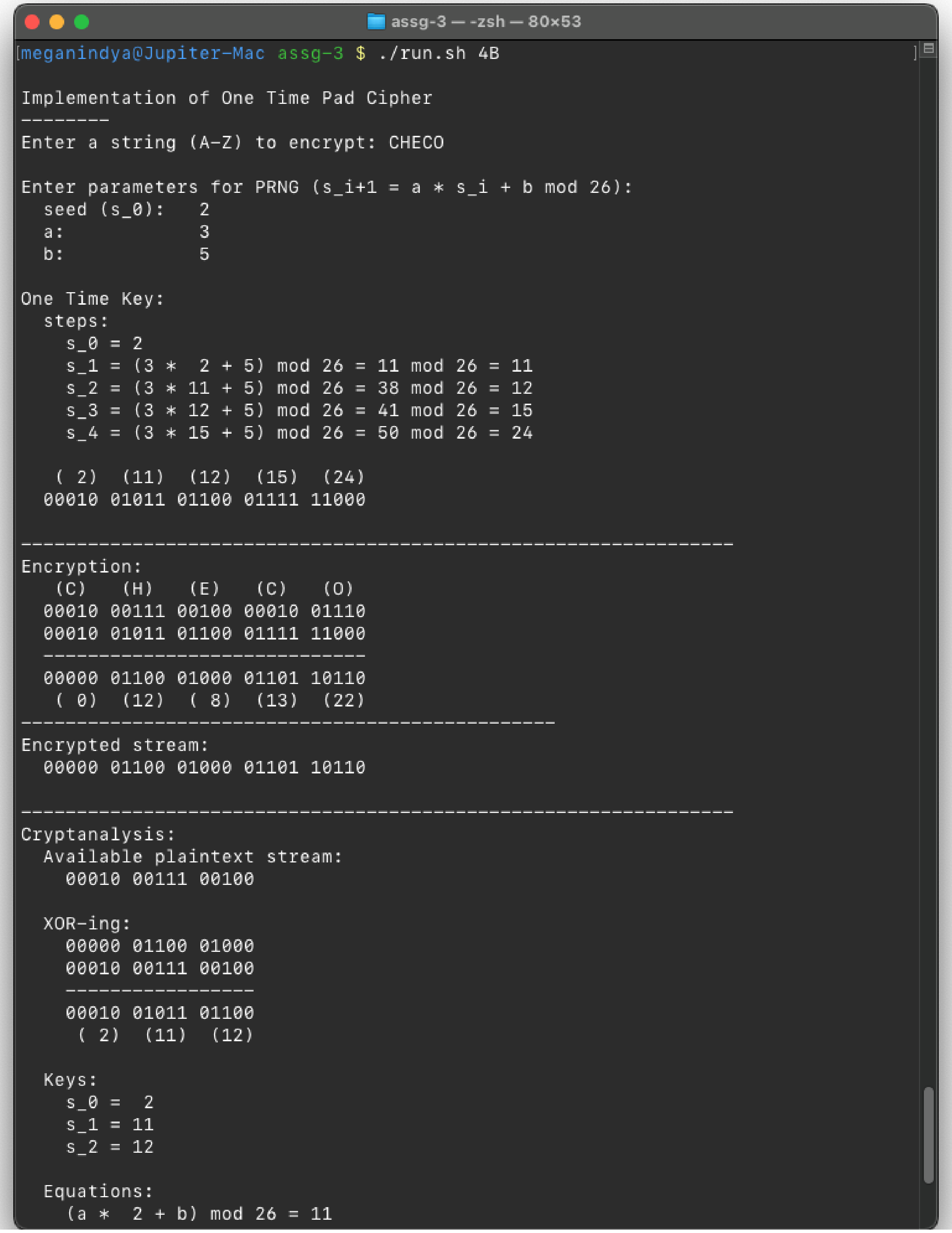


```
        }
        if (i != strlen(s) - 1)
        {
            printf(
                "    s_%d = (%d * %2d + %d) mod %d = %2d mod %d = %2d\n",
                i + 1,
                a,
                random_s,
                b,
                MOD,
                a * random_s + b,
                MOD,
                mod_26(a * random_s + b));
        }
        random_s = mod_26(a * random_s + b);
    }
    printf("\n  ");
    for (int i = 0; i < strlen(s); i++)
    {
        printf(" (%2d) ", k_n[i]);
    }
    printf("\n  ");
    for (int i = 0; i < strlen(s); i++)
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", k[j]);
        }
        printf(" ");
    }
    printf("\n");

    printf("\n----------------------------------------------------------------
\nEncryption:\n");
    encrypt(s, k);
    printf("------------------------------------------------\nEncrypted stream:\n  ");
    for (int i = 0; i < strlen(s); i++)
    {
        char bin[5];
        int_to_binary(s[i] - 'A', bin);
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", bin[j - (i * 5)]);
        }
        printf(" ");
    }
    printf("\n");

    printf("\n----------------------------------------------------------------
\nCryptanalysis:\n");
    printf("  Available plaintext stream:\n    ");
    for (int i = 0; i < 3; i++)
```

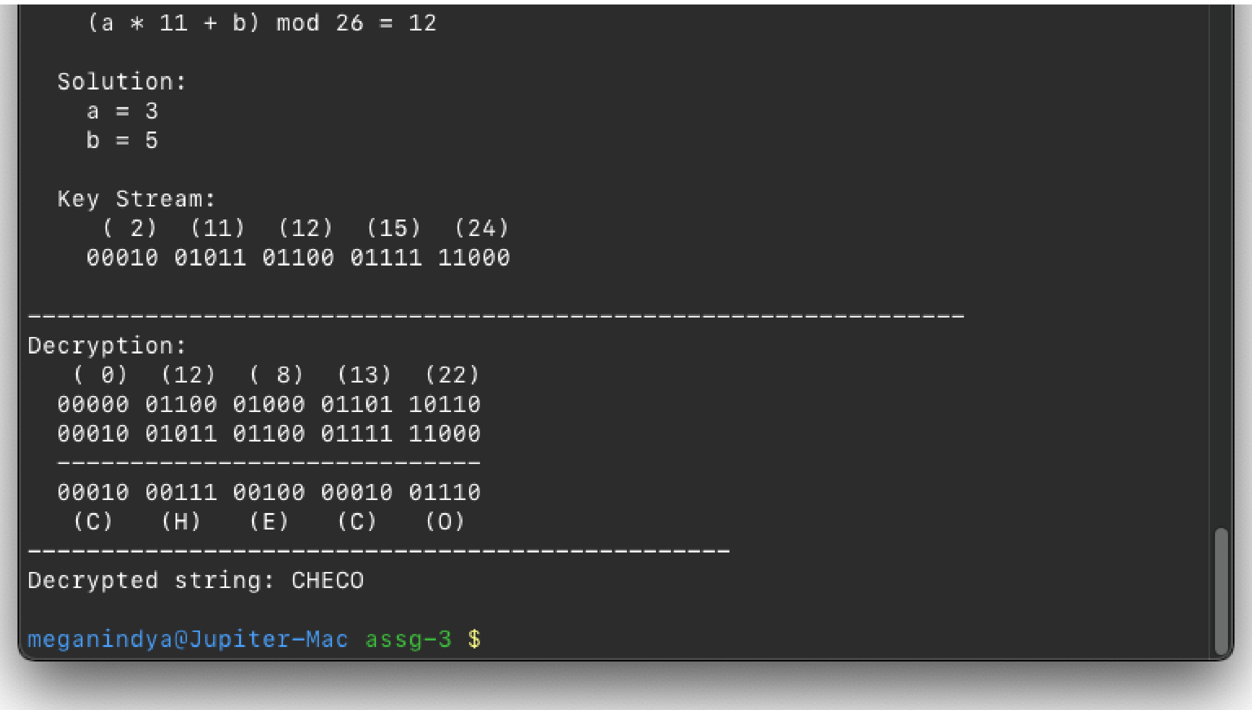


```
    {
        for (int j = i * 5; j < (i + 1) * 5; j++)
        {
            printf("%c", bin_s[j]);
        }
        printf(" ");
    }
    printf("\n\n");
    char k_g[256];
    cryptanalyze(s, bin_s, k_g);

    printf("\n----------------------------------------------------------------
\nDecryption:\n");
    decrypt(s, k_g);
    printf("------------------------------------------------\nDecrypted string: %s\n", s);
    printf("\n");
}
```

Sample run



```
[meganindya@Jupiter-Mac assg-3 $ ./run.sh 4B                                  ]

Implementation of One Time Pad Cipher
--------
Enter a string (A-Z) to encrypt: CHECO

Enter parameters for PRNG (s_i+1 = a * s_i + b mod 26):
  seed (s_0):   2
  a:            3
  b:            5

One Time Key:
  steps:
    s_0 = 2
    s_1 = (3 *  2 + 5) mod 26 = 11 mod 26 = 11
    s_2 = (3 * 11 + 5) mod 26 = 38 mod 26 = 12
    s_3 = (3 * 12 + 5) mod 26 = 41 mod 26 = 15
    s_4 = (3 * 15 + 5) mod 26 = 50 mod 26 = 24

   ( 2)  (11)  (12)  (15)  (24)
  00010 01011 01100 01111 11000

----------------------------------------------------------------
Encryption:
   (C)   (H)   (E)   (C)   (O)
  00010 00111 00100 00010 01110
  00010 01011 01100 01111 11000
  -----------------------------
  00000 01100 01000 01101 10110
   ( 0)  (12)  ( 8)  (13)  (22)
------------------------------------------------
Encrypted stream:
  00000 01100 01000 01101 10110

----------------------------------------------------------------
Cryptanalysis:
  Available plaintext stream:
    00010 00111 00100

  XOR-ing:
    00000 01100 01000
    00010 00111 00100
    -----------------
    00010 01011 01100
     ( 2)  (11)  (12)

  Keys:
    s_0 =  2
    s_1 = 11
    s_2 = 12

  Equations:
    (a *  2 + b) mod 26 = 11
```



```
    (a * 11 + b) mod 26 = 12

  Solution:
    a = 3
    b = 5

  Key Stream:
     ( 2)  (11)  (12)  (15)  (24)
    00010 01011 01100 01111 11000

----------------------------------------------------------------
Decryption:
   ( 0)  (12)  ( 8)  (13)  (22)
  00000 01100 01000 01101 10110
  00010 01011 01100 01111 11000
  -----------------------------
  00010 00111 00100 00010 01110
   (C)   (H)   (E)   (C)   (O)
------------------------------------------------
Decrypted string: CHECO

meganindya@Jupiter-Mac assg-3 $
```