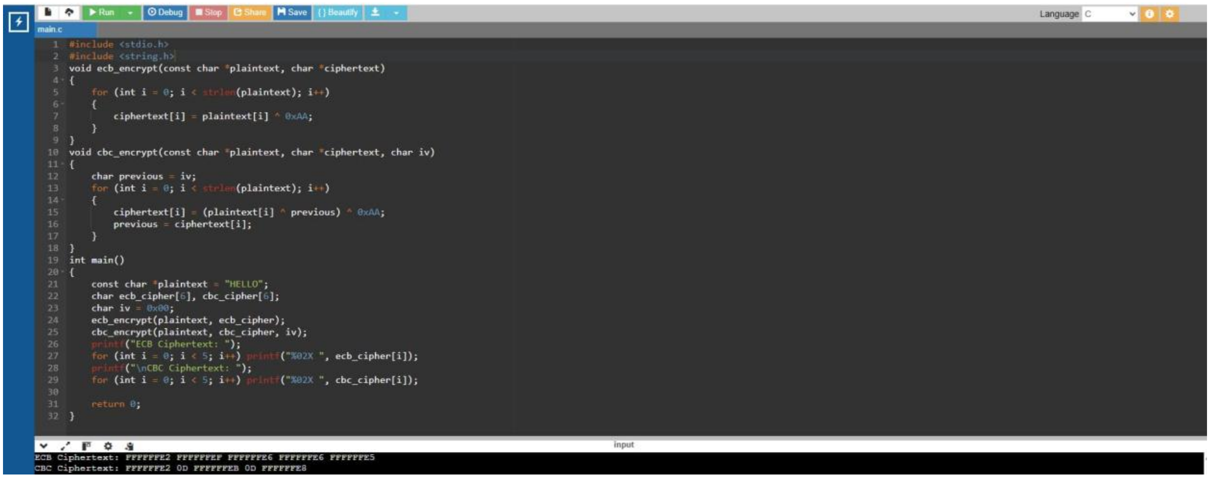
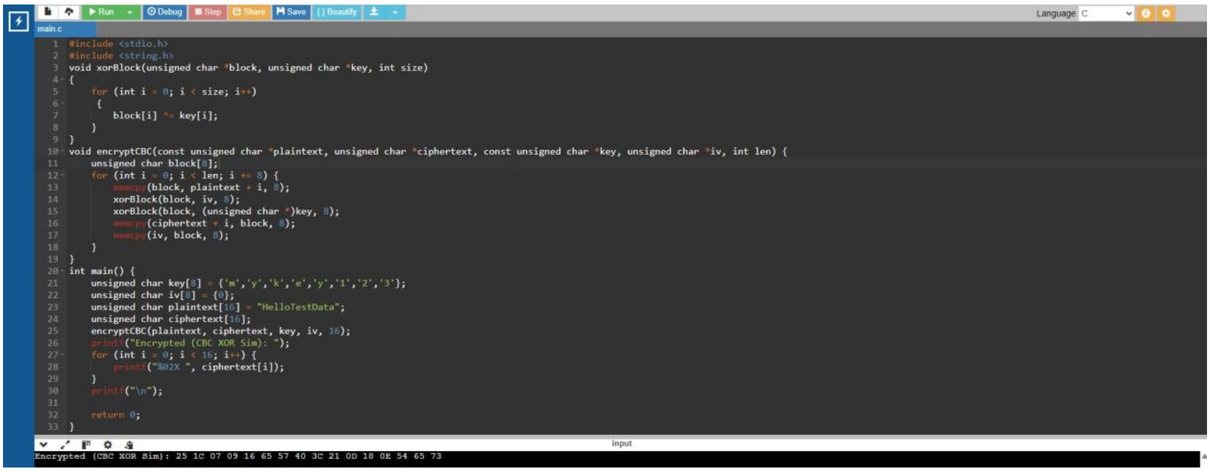


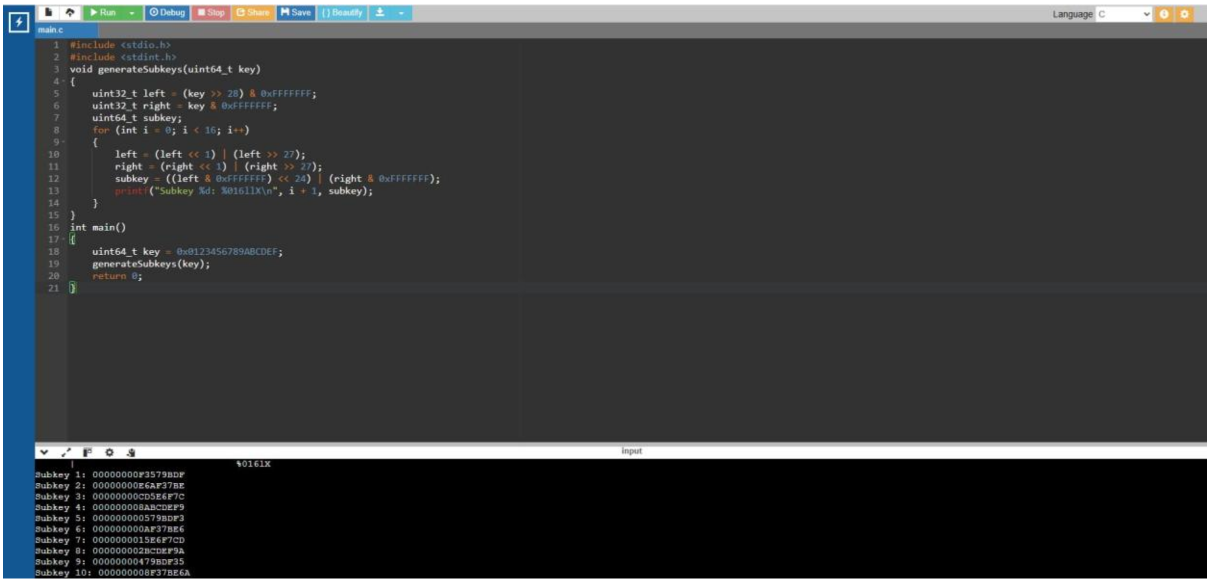
```c
#include <stdio.h>
#include <string.h>
void ecb_encrypt(const char *plaintext, char *ciphertext)
{
    for (int i = 0; i < strlen(plaintext); i++)
    {
        ciphertext[i] = plaintext[i] ^ 0xAA;
    }
}
void cbc_encrypt(const char *plaintext, char *ciphertext, char iv)
{
    char previous = iv;
    for (int i = 0; i < strlen(plaintext); i++)
    {
        ciphertext[i] = (plaintext[i] ^ previous) ^ 0xAA;
        previous = ciphertext[i];
    }
}
int main()
{
    const char *plaintext = "HELLO";
    char ecb_cipher[6], cbc_cipher[6];
    char iv = 0x00;
    ecb_encrypt(plaintext, ecb_cipher);
    cbc_encrypt(plaintext, cbc_cipher, iv);
    printf("ECB Ciphertext: ");
    for (int i = 0; i < 5; i++) printf("%02X ", ecb_cipher[i]);
    printf("\nCBC Ciphertext: ");
    for (int i = 0; i < 5; i++) printf("%02X ", cbc_cipher[i]);

    return 0;
}
```

```
ECB Ciphertext: FFFFFFE2 FFFFFFEF FFFFFFE6 FFFFFFE6 FFFFFFE5
CBC Ciphertext: FFFFFFE2 0D FFFFFFEB 0D FFFFFFE8
```



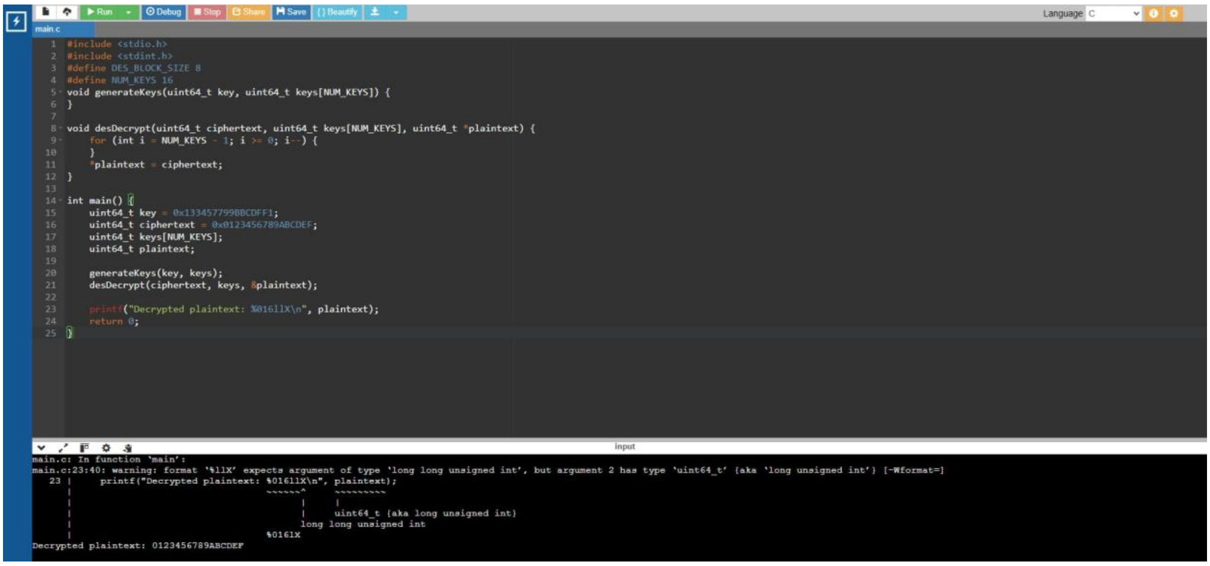
```c
#include <stdio.h>
#include <string.h>
void xorBlock(unsigned char *block, unsigned char *key, int size)
{
    for (int i = 0; i < size; i++)
     {
        block[i] ^= key[i];
    }
}
void encryptCBC(const unsigned char *plaintext, unsigned char *ciphertext, const unsigned char *key, unsigned char *iv, int len) {
    unsigned char block[8];
    for (int i = 0; i < len; i += 8) {
        memcpy(block, plaintext + i, 8);
        xorBlock(block, iv, 8);
        xorBlock(block, (unsigned char *)key, 8);
        memcpy(ciphertext + i, block, 8);
        memcpy(iv, block, 8);
    }
}
int main() {
    unsigned char key[8] = {'m','y','k','e','y','1','2','3'};
    unsigned char iv[8] = {0};
    unsigned char plaintext[16] = "HelloTestData";
    unsigned char ciphertext[16];
    encryptCBC(plaintext, ciphertext, key, iv, 16);
    printf("Encrypted (CBC XOR Sim): ");
    for (int i = 0; i < 16; i++) {
        printf("%02X ", ciphertext[i]);
    }
    printf("\n");

    return 0;
}
```

```
Encrypted (CBC XOR Sim): 25 1C 07 09 16 65 57 40 3C 21 0D 18 0E 54 65 73
```



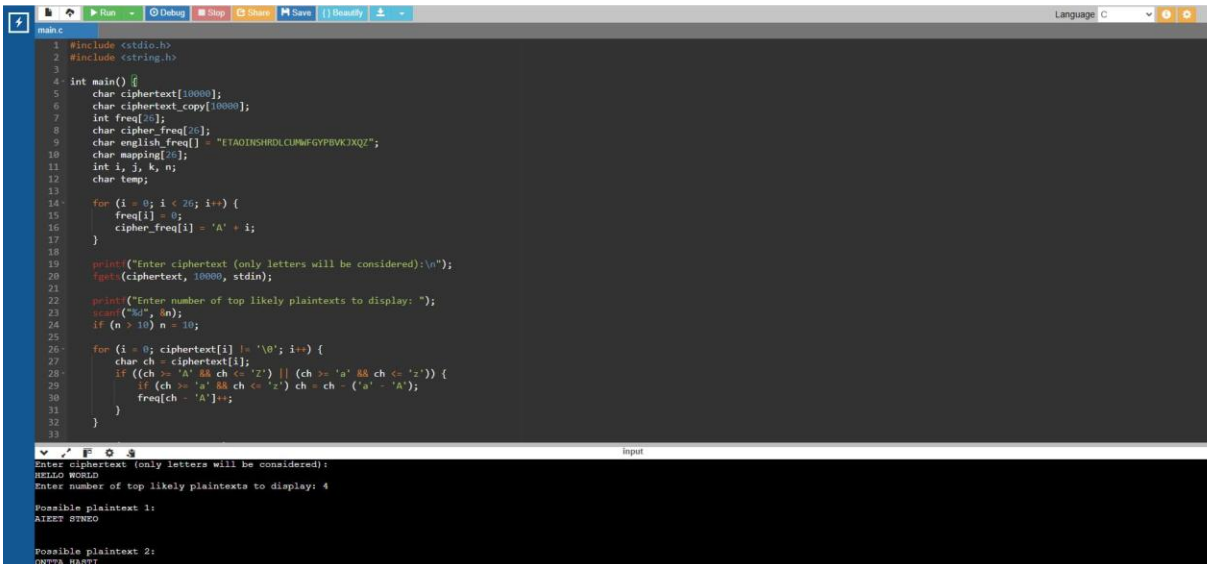
```c
#include <stdio.h>
#include <stdint.h>
void generateSubkeys(uint64_t key)
{
    uint32_t left = (key >> 28) & 0xFFFFFFF;
    uint32_t right = key & 0xFFFFFFF;
    uint64_t subkey;
    for (int i = 0; i < 16; i++)
    {
        left = (left << 1) | (left >> 27);
        right = (right << 1) | (right >> 27);
        subkey = ((left & 0xFFFFFFF) << 24) | (right & 0xFFFFFFF);
        printf("Subkey %d: %016llX\n", i + 1, subkey);
    }
}
int main()
{
    uint64_t key = 0x0123456789ABCDEF;
    generateSubkeys(key);
    return 0;
}
```

```
                        %0161X
Subkey 1: 00000000F3579BDF
Subkey 2: 00000000E6AF37BE
Subkey 3: 00000000CD5E6F7C
Subkey 4: 000000008ABCDEF9
Subkey 5: 000000000579BDF3
Subkey 6: 000000000AF37BE6
Subkey 7: 0000000015E6F7CD
Subkey 8: 000000002BCDEF9A
Subkey 9: 00000000479BDF35
Subkey 10: 000000008F37BE6A
```



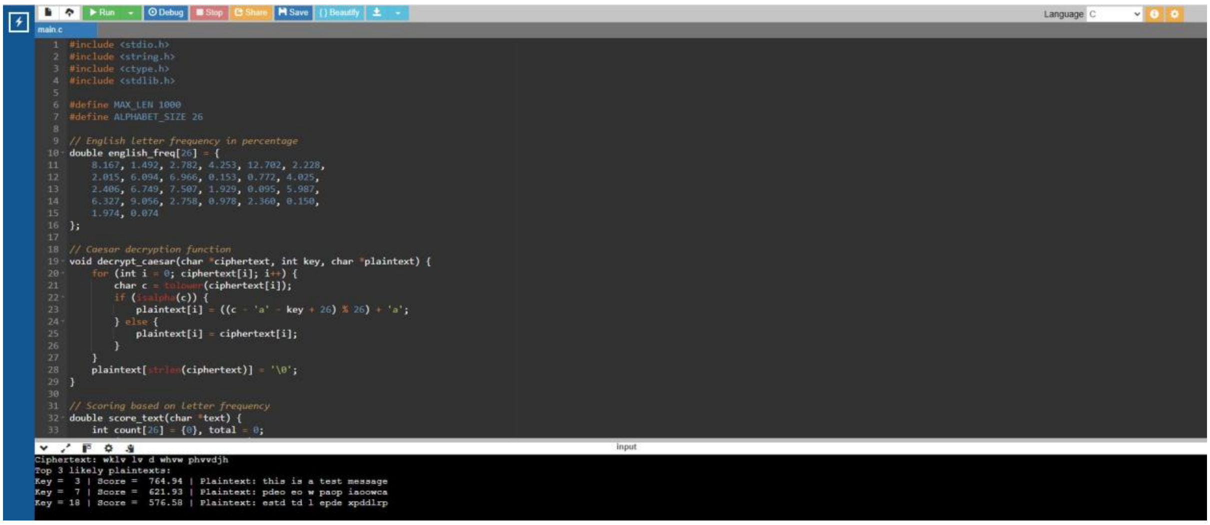
```c
#include <stdio.h>
#include <stdint.h>
#define DES_BLOCK_SIZE 8
#define NUM_KEYS 16
void generateKeys(uint64_t key, uint64_t keys[NUM_KEYS]) {
}

void desDecrypt(uint64_t ciphertext, uint64_t keys[NUM_KEYS], uint64_t *plaintext) {
    for (int i = NUM_KEYS - 1; i >= 0; i--) {
    }
    *plaintext = ciphertext;
}

int main() {
    uint64_t key = 0x133457799BBCDFF1;
    uint64_t ciphertext = 0x0123456789ABCDEF;
    uint64_t keys[NUM_KEYS];
    uint64_t plaintext;

    generateKeys(key, keys);
    desDecrypt(ciphertext, keys, &plaintext);

    printf("Decrypted plaintext: %016llX\n", plaintext);
    return 0;
}
```

```
main.c: In function ‘main’:
main.c:23:40: warning: format ‘%llX’ expects argument of type ‘long long unsigned int’, but argument 2 has type ‘uint64_t’ {aka ‘long unsigned int’} [-Wformat=]
   23 |     printf("Decrypted plaintext: %016llX\n", plaintext);
      |                                  ~~~~~~^     ~~~~~~~~~
      |                                        |     |
      |                                        |     uint64_t {aka long unsigned int}
      |                                        long long unsigned int
      |                                  %016lx
Decrypted plaintext: 0123456789ABCDEF
```



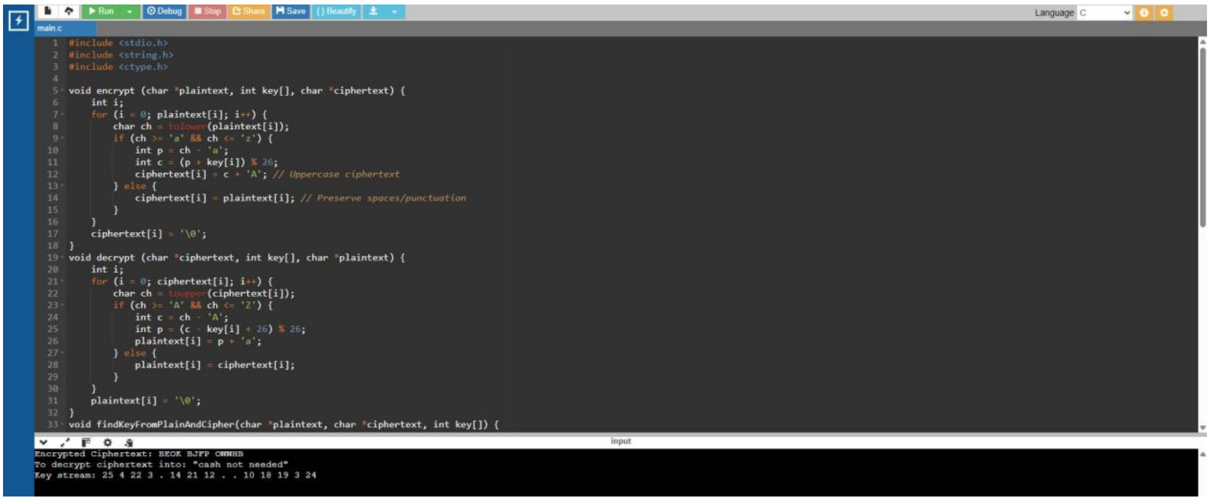
```c
#include <stdio.h>
#include <string.h>

int main() {
    char ciphertext[10000];
    char ciphertext_copy[10000];
    int freq[26];
    char cipher_freq[26];
    char english_freq[] = "ETAOINSHRDLCUMWFGYPBVKJXQZ";
    char mapping[26];
    int i, j, k, n;
    char temp;

    for (i = 0; i < 26; i++) {
        freq[i] = 0;
        cipher_freq[i] = 'A' + i;
    }

    printf("Enter ciphertext (only letters will be considered):\n");
    fgets(ciphertext, 10000, stdin);

    printf("Enter number of top likely plaintexts to display: ");
    scanf("%d", &n);
    if (n > 10) n = 10;

    for (i = 0; ciphertext[i] != '\0'; i++) {
        char ch = ciphertext[i];
        if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z')) {
            if (ch >= 'a' && ch <= 'z') ch = ch - ('a' - 'A');
            freq[ch - 'A']++;
        }
    }

```

```
Enter ciphertext (only letters will be considered):
HELLO WORLD
Enter number of top likely plaintexts to display: 4

Possible plaintext 1:
AIEET STNEO


Possible plaintext 2:
ONTTA HASTI
```



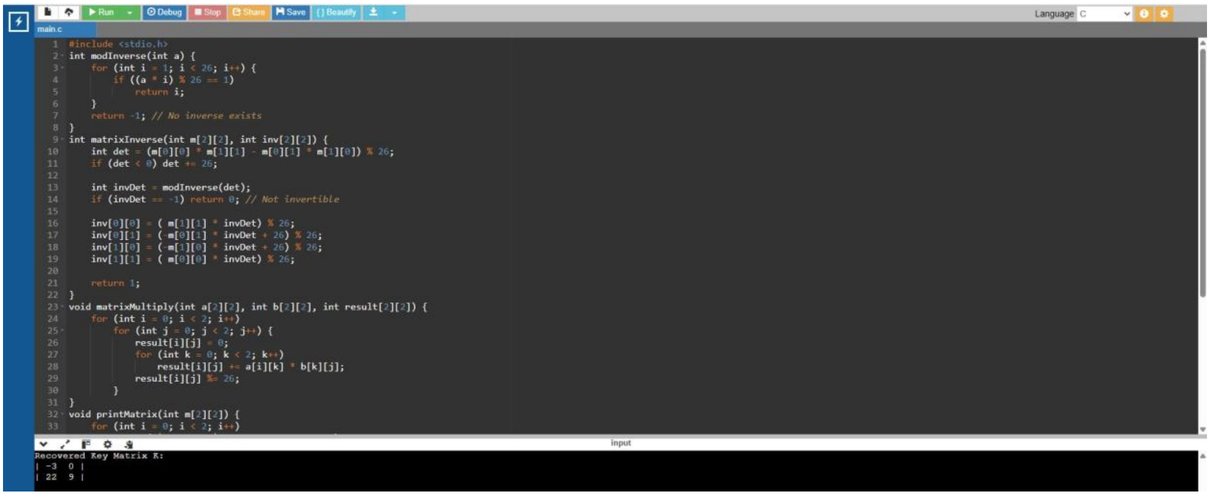
```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

#define MAX_LEN 1000
#define ALPHABET_SIZE 26

// English letter frequency in percentage
double english_freq[26] = {
    8.167, 1.492, 2.782, 4.253, 12.702, 2.228,
    2.015, 6.094, 6.966, 0.153, 0.772, 4.025,
    2.406, 6.749, 7.507, 1.929, 0.095, 5.987,
    6.327, 9.056, 2.758, 0.978, 2.360, 0.150,
    1.974, 0.074
};

// Caesar decryption function
void decrypt_caesar(char *ciphertext, int key, char *plaintext) {
    for (int i = 0; ciphertext[i]; i++) {
        char c = tolower(ciphertext[i]);
        if (isalpha(c)) {
            plaintext[i] = ((c - 'a' - key + 26) % 26) + 'a';
        } else {
            plaintext[i] = ciphertext[i];
        }
    }
    plaintext[strlen(ciphertext)] = '\0';
}

// Scoring based on letter frequency
double score_text(char *text) {
    int count[26] = {0}, total = 0;
```

```
Ciphertext: wklv lv d whvw phvvdjh
Top 3 likely plaintexts:
Key =  3 | Score =  764.94 | Plaintext: this is a test message
Key =  7 | Score =  621.93 | Plaintext: pdeo eo w paop iaoowca
Key = 18 | Score =  576.58 | Plaintext: estd td l epde xpddlrp
```



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

void encrypt (char *plaintext, int key[], char *ciphertext) {
    int i;
    for (i = 0; plaintext[i]; i++) {
        char ch = tolower(plaintext[i]);
        if (ch >= 'a' && ch <= 'z') {
            int p = ch - 'a';
            int c = (p + key[i]) % 26;
            ciphertext[i] = c + 'A'; // Uppercase ciphertext
        } else {
            ciphertext[i] = plaintext[i]; // Preserve spaces/punctuation
        }
    }
    ciphertext[i] = '\0';
}
void decrypt (char *ciphertext, int key[], char *plaintext) {
    int i;
    for (i = 0; ciphertext[i]; i++) {
        char ch = toupper(ciphertext[i]);
        if (ch >= 'A' && ch <= 'Z') {
            int c = ch - 'A';
            int p = (c - key[i] + 26) % 26;
            plaintext[i] = p + 'a';
        } else {
            plaintext[i] = ciphertext[i];
        }
    }
    plaintext[i] = '\0';
}
void findKeyFromPlainAndCipher(char *plaintext, char *ciphertext, int key[]) {
```

```
Encrypted Ciphertext: BEOK BJFP OWWHB
To decrypt ciphertext into: "cash not needed"
Key stream: 25 4 22 3 . 14 21 12 . . 10 18 19 3 24
```



```c
#include <stdio.h>
int modInverse(int a) {
    for (int i = 1; i < 26; i++) {
        if ((a * i) % 26 == 1)
            return i;
    }
    return -1; // No inverse exists
}
int matrixInverse(int m[2][2], int inv[2][2]) {
    int det = (m[0][0] * m[1][1] - m[0][1] * m[1][0]) % 26;
    if (det < 0) det += 26;

    int invDet = modInverse(det);
    if (invDet == -1) return 0; // Not invertible

    inv[0][0] = ( m[1][1] * invDet) % 26;
    inv[0][1] = (-m[0][1] * invDet + 26) % 26;
    inv[1][0] = (-m[1][0] * invDet + 26) % 26;
    inv[1][1] = ( m[0][0] * invDet) % 26;

    return 1;
}
void matrixMultiply(int a[2][2], int b[2][2], int result[2][2]) {
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++) {
            result[i][j] = 0;
            for (int k = 0; k < 2; k++)
                result[i][j] += a[i][k] * b[k][j];
            result[i][j] %= 26;
        }
}
void printMatrix(int m[2][2]) {
    for (int i = 0; i < 2; i++)
```

```
Recovered Key Matrix K:
| -3  0 |
| 22  9 |
```