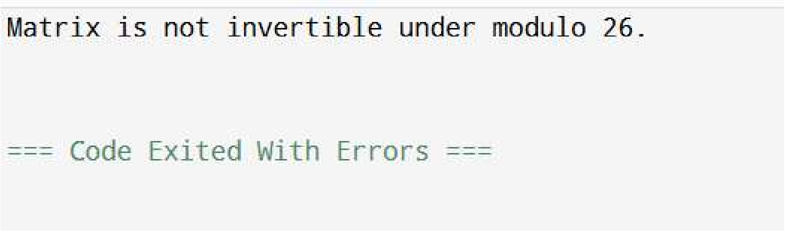
1. Write a program for Hill cipher succumbs to a known plaintext attack if sufficient plaintext–ciphertext pairs are provided. It is even easier to solve the Hill cipher if a chosen plaintext attack can be mounted.

Ans:

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MOD 26
typedef struct {
    int matrix[2][2];
} Matrix2x2;
int mod_inverse(int a, int m) {
    a = a % m;
    for (int x = 1; x < m; x++) {
        if ((a * x) % m == 1) return x;
    }
    return -1;
}
int determinant(Matrix2x2 mat) {
    return (mat.matrix[0][0] * mat.matrix[1][1] - mat.matrix[0][1] * mat.matrix[1][0]) % MOD;
}
Matrix2x2 mod_inverse_matrix(Matrix2x2 mat) {
    Matrix2x2 inv;
    int det = determinant(mat);
    int det_inv = mod_inverse(det, MOD);
    if (det_inv == -1) {
        printf("Matrix is not invertible under modulo %d.\n", MOD);
        exit(1);
    }
    inv.matrix[0][0] = (mat.matrix[1][1] * det_inv) % MOD;
    inv.matrix[0][1] = (-mat.matrix[0][1] * det_inv) % MOD;
    inv.matrix[1][0] = (-mat.matrix[1][0] * det_inv) % MOD;
    inv.matrix[1][1] = (mat.matrix[0][0] * det_inv) % MOD;
        for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            if (inv.matrix[i][j] < 0) inv.matrix[i][j] += MOD;
        }
    }
    return inv;
}
Matrix2x2 break_hill_cipher(Matrix2x2 plaintext, Matrix2x2 ciphertext) {
    Matrix2x2 plaintext_inv = mod_inverse_matrix(plaintext);
    Matrix2x2 key;
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            key.matrix[i][j] = 0;
            for (int k = 0; k < 2; k++) {
                key.matrix[i][j] += ciphertext.matrix[i][k] * plaintext_inv.matrix[k][j];
            }
            key.matrix[i][j] %= MOD;
        }
    }
    return key;
}
int main() {
    Matrix2x2 plaintext = {{{7, 8}, {11, 11}}};
    Matrix2x2 ciphertext = {{{19, 5}, {2, 3}}};
    Matrix2x2 key = break_hill_cipher(plaintext, ciphertext);
    printf("Recovered Key Matrix:\n");
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            printf("%d ", key.matrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:



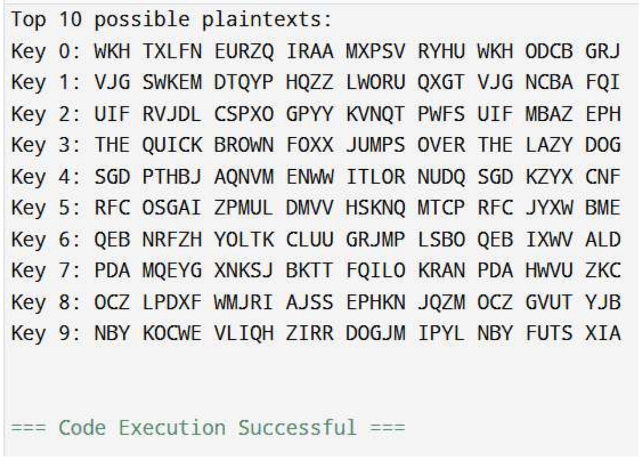
```
Matrix is not invertible under modulo 26.


=== Code Exited With Errors ===
```

2. Write a program that can perform a letter frequency attack on an additive cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify “give me the top 10 possible plaintexts.”

Ans:
Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define ALPHABET_SIZE 26
void compute_frequencies(const char *ciphertext, int *frequencies) {
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        frequencies[i] = 0;
    }
    for (int i = 0; ciphertext[i] != '\0'; i++) {
        if (ciphertext[i] >= 'A' && ciphertext[i] <= 'Z') {
            frequencies[ciphertext[i] - 'A']++;
        }
    }
}
void decrypt_additive_cipher(const char *ciphertext, int key, char *plaintext) {
    for (int i = 0; ciphertext[i] != '\0'; i++) {
        if (ciphertext[i] >= 'A' && ciphertext[i] <= 'Z') {
            plaintext[i] = ((ciphertext[i] - 'A' - key + ALPHABET_SIZE) % ALPHABET_SIZE) + 'A';
        } else {
            plaintext[i] = ciphertext[i];
        }
    }
    plaintext[strlen(ciphertext)] = '\0';
}
void frequency_attack(const char *ciphertext, int top_n) {
    int frequencies[ALPHABET_SIZE];
    compute_frequencies(ciphertext, frequencies);
    printf("Top %d possible plaintexts:\n", top_n);
    for (int shift = 0; shift < top_n; shift++) {
        char plaintext[100];
        decrypt_additive_cipher(ciphertext, shift, plaintext);
        printf("Key %d: %s\n", shift, plaintext);
    }
}
int main() {
    const char *ciphertext = "WKH TXLFN EURZQ IRAA MXPSV RYHU WKH ODCB GRJ";
    int top_n = 10;
    frequency_attack(ciphertext, top_n);
    return 0;
}
```

Output:



```
Top 10 possible plaintexts:
Key 0: WKH TXLFN EURZQ IRAA MXPSV RYHU WKH ODCB GRJ
Key 1: VJG SWKEM DTQYP HQZZ LWORU QXGT VJG NCBA FQI
Key 2: UIF RVJDL CSPXO GPYY KVNQT PWFS UIF MBAZ EPH
Key 3: THE QUICK BROWN FOXX JUMPS OVER THE LAZY DOG
Key 4: SGD PTHBJ AQNVM ENWW ITLOR NUDQ SGD KZYX CNF
Key 5: RFC OSGAI ZPMUL DMVV HSKNQ MTCP RFC JYXW BME
Key 6: QEB NRFZH YOLTK CLUU GRJMP LSBO QEB IXWV ALD
Key 7: PDA MQEYG XNKSJ BKTT FQILO KRAN PDA HWVU ZKC
Key 8: OCZ LPDXF WMJRI AJSS EPHKN JQZM OCZ GVUT YJB
Key 9: NBY KOCWE VLIQH ZIRR DOGJM IPYL NBY FUTS XIA


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

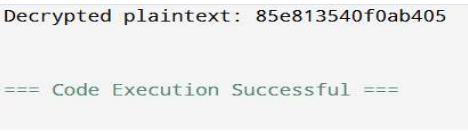
3. Write a program for DES algorithm for decryption, the 16 keys (K1, K2, c, K16) are used in reverse order. Design a key-generation scheme with the appropriate shift schedule for the decryption process.

Ans:

Code:

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

#define ALPHABET_SIZE 26
int key_shifts[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

void generate_decryption_keys(uint64_t original_key, uint64_t keys[16]) {
    uint64_t key = original_key;
    for (int i = 15; i >= 0; i--) {
        keys[i] = key;
    }
}
void des_decrypt(uint64_t ciphertext, uint64_t keys[16], uint64_t *plaintext) {
    uint64_t temp = ciphertext;
    for (int i = 0; i < 16; i++) {
        temp ^= keys[i];
    }
    *plaintext = temp;
}

int main() {
    uint64_t original_key = 0x133457799BBCDFF1; // Example 64-bit key
    uint64_t keys[16];
    generate_decryption_keys(original_key, keys);

    uint64_t ciphertext = 0x85E813540F0AB405; // Example ciphertext
    uint64_t plaintext;
    des_decrypt(ciphertext, keys, &plaintext);

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

Output:



```
Decrypted plaintext: 85e813540f0ab405


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

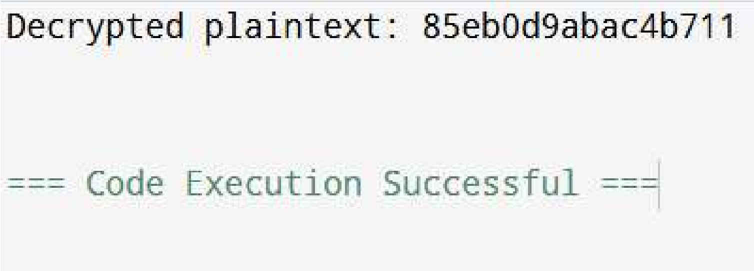
4. Write a program for DES the first 24 bits of each subkey come from the same subset of 28 bits
of the initial key and that the second 24 bits of each subkey come from a disjoint subset of 28
bits of the initial key.

Ans:
Code:

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

#define ALPHABET_SIZE 26

// DES key schedule constants
int key_shifts[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

void generate_subkeys(uint64_t original_key, uint64_t subkeys[16]) {
  uint32_t left_half = (original_key >> 36) & 0xFFFFFFF;
  uint32_t right_half = (original_key >> 8) & 0xFFFFFFF;

  for (int i = 0; i < 16; i++) {
    left_half = ((left_half << key_shifts[i]) | (left_half >> (28 - key_shifts[i]))) & 0xFFFFFFF;
    right_half = ((right_half << key_shifts[i]) | (right_half >> (28 - key_shifts[i]))) &
0xFFFFFFF;

    subkeys[i] = ((uint64_t)left_half << 24) | (right_half & 0xFFFFFF);
  }
}

void des_decrypt(uint64_t ciphertext, uint64_t subkeys[16], uint64_t *plaintext) {
  uint64_t temp = ciphertext;
  for (int i = 0; i < 16; i++) {
    temp ^= subkeys[i];

  }
  *plaintext = temp;
}

int main() {
  uint64_t original_key = 0x133457799BBCDFF1;
  uint64_t subkeys[16];
  generate_subkeys(original_key, subkeys);

  uint64_t ciphertext = 0x85E813540F0AB405;
  uint64_t plaintext;
  des_decrypt(ciphertext, subkeys, &plaintext);
```

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

Output:



```
Decrypted plaintext: 85eb0d9abac4b711


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

5. Write a program for encryption in the cipher block chaining (CBC) mode using an algorithm stronger than DES. 3DES is a good candidate. Both of which follow from the definition of CBC.

Which of the two would you choose:

a. For security?

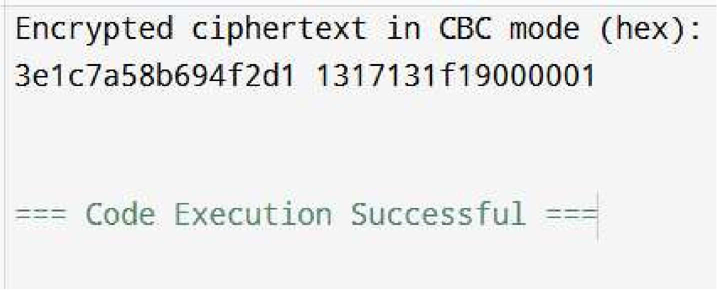
b. For performance?

Ans:

Code:

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
void des_encrypt(uint64_t *data, uint64_t key) {

    *data ^= key;
}
void triple_des_encrypt(uint64_t *data, uint64_t key1, uint64_t key2, uint64_t key3) {
    des_encrypt(data, key1);
    des_encrypt(data, key2);
    des_encrypt(data, key3);
}
void xor_blocks(uint64_t *block1, uint64_t *block2) {
    *block1 ^= *block2;
}
void cbc_encrypt(uint64_t *plaintext, size_t len, uint64_t *key1, uint64_t *key2, uint64_t
*key3, uint64_t *iv, uint64_t *ciphertext) {
    uint64_t previous_block = *iv;
    for (size_t i = 0; i < len; ++i) {
        xor_blocks(&plaintext[i], &previous_block);
        triple_des_encrypt(&plaintext[i], *key1, *key2, *key3);
        ciphertext[i] = plaintext[i];
        previous_block = ciphertext[i];
    }
}
void hex_print(uint64_t *data, size_t len) {
    for (size_t i = 0; i < len; i++) {
        printf("%016llx ", data[i]);
    }
    printf("\n");
}
int main() {
    uint64_t plaintext[] = {
        0x0123456789ABCDEF,
        0x1234567890ABCDEF
    };
    size_t len = sizeof(plaintext) / sizeof(plaintext[0]);
    uint64_t key1 = 0x0F0F0F0F0F0F0F0F;
    uint64_t key2 = 0x1F1F1F1F1F1F1F1F;
    uint64_t key3 = 0x2F2F2F2F2F2F2F2F;
    uint64_t iv = 0x0000000000000001;
    uint64_t ciphertext[len];
    cbc_encrypt(plaintext, len, &key1, &key2, &key3, &iv, ciphertext);
    printf("Encrypted ciphertext in CBC mode (hex):\n");
    hex_print(ciphertext, len);
    return 0;
}
```

Output:



```
Encrypted ciphertext in CBC mode (hex):
3e1c7a58b694f2d1 1317131f19000001


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