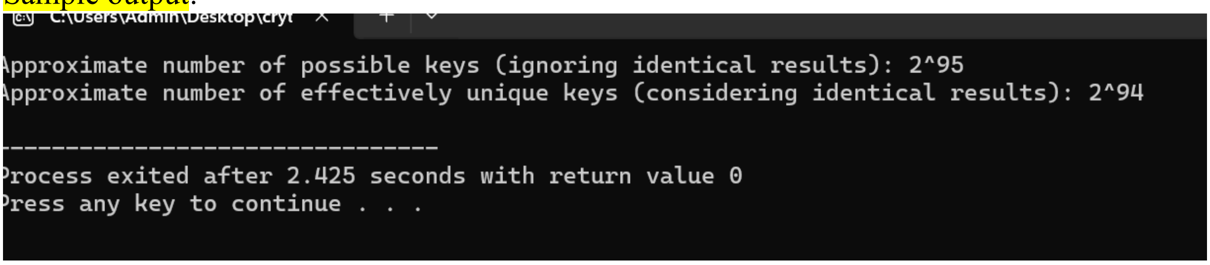
1.Write a C program for possible keys does the Playfair cipher have? Ignore the fact that some keys might produce identical encryption results. Express your answer as an approximate power of 2.

Program:

```
#include <stdio.h>
#include <math.h>
double stirlingApproximation(int n) {
   return n * log2(n) - n + log2(2 * M_PI * n) / 2;
}
int main() {
   int n = 25;
   double power_of_2_without_symmetry = stirlingApproximation(n);
   double power_of_2_with_symmetry = power_of_2_without_symmetry - 1;
   printf("Approximate number of possible keys (ignoring identical results): 2^%.0f\n",
power_of_2_without_symmetry);
   printf("Approximate number of effectively unique keys (considering identical
results): 2^%.0f\n", power_of_2_with_symmetry);
   return 0;
}
```

Sample output:



```
Approximate number of possible keys (ignoring identical results): 2^95
Approximate number of effectively unique keys (considering identical results): 2^94

--------------------------------
Process exited after 2.425 seconds with return value 0
Press any key to continue . . .
```

12. a. Write a C program to Encrypt the message "meet me at the usual place at ten rather than eight oclock" using the Hill cipher with the key.

(9 4

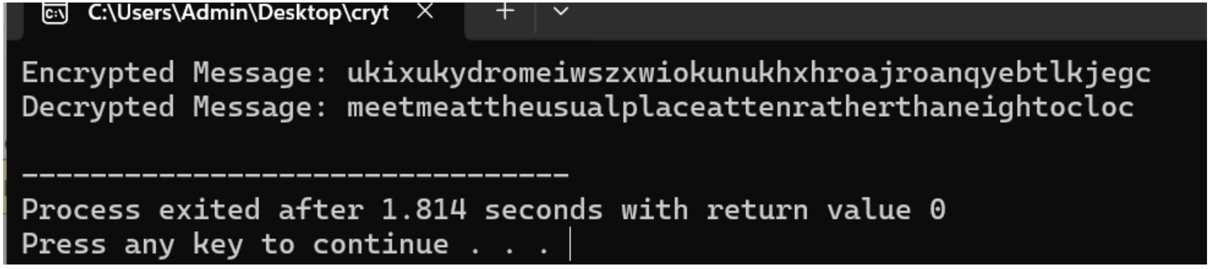
5 7)

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 2
int modInverse(int a, int m) {
   for (int x = 1; x < m; x++) {
      if ((a * x) % m == 1)
         return x;
   }
   return -1;
}
int determinant(int matrix[SIZE][SIZE]) {
   return (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]);
```

```
}
int modMatrixInverse(int matrix[SIZE][SIZE], int inverse[SIZE][SIZE]) {
    int det = determinant(matrix);
    int mod_det = modInverse(det, 26);

    if (mod_det == -1) {
        return 0;
    }

    inverse[0][0] = (matrix[1][1] * mod_det) % 26;
    inverse[0][1] = (-matrix[0][1] * mod_det) % 26;
    inverse[1][0] = (-matrix[1][0] * mod_det) % 26;
    inverse[1][1] = (matrix[0][0] * mod_det) % 26;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (inverse[i][j] < 0) {
                inverse[i][j] += 26;
            }
        }
    }

    return 1;
}
void encrypt(char text[], int key[SIZE][SIZE], char cipher[]) {
    int len = strlen(text);
    int pairs = len / 2;
    if (len % 2 != 0) {
        text[len] = 'X';
        text[len + 1] = '\0';
        pairs++;
    }
    for (int i = 0; i < pairs; i++) {
        int x1 = text[2 * i] - 'a';
        int x2 = text[2 * i + 1] - 'a';
        int y1 = (key[0][0] * x1 + key[0][1] * x2) % 26;
        int y2 = (key[1][0] * x1 + key[1][1] * x2) % 26;
        cipher[2 * i] = y1 + 'a';
        cipher[2 * i + 1] = y2 + 'a';
    }

    cipher[len] = '\0';
}
void decrypt(char cipher[], int key[SIZE][SIZE], char plain[]) {
    int len = strlen(cipher);
    int pairs = len / 2;
    int inverse[SIZE][SIZE];
    if (!modMatrixInverse(key, inverse)) {
        printf("Inverse matrix does not exist\n");
        return;
    }

    for (int i = 0; i < pairs; i++) {
        int y1 = cipher[2 * i] - 'a';
        int y2 = cipher[2 * i + 1] - 'a';
        int x1 = (inverse[0][0] * y1 + inverse[0][1] * y2) % 26;
        int x2 = (inverse[1][0] * y1 + inverse[1][1] * y2) % 26;
        plain[2 * i] = x1 + 'a';
        plain[2 * i + 1] = x2 + 'a';
    }

    plain[len] = '\0';
}
int main() {
    char text[] = "meetmeattheusualplaceattenratherthaneightoclock";
    char cipher[100];
    char decrypted[100];
    int key[SIZE][SIZE] = {{9, 4}, {5, 7}};
    encrypt(text, key, cipher);
    printf("Encrypted Message: %s\n", cipher);
    decrypt(cipher, key, decrypted);
    printf("Decrypted Message: %s\n", decrypted);
    return 0;
}
```

Sample output:



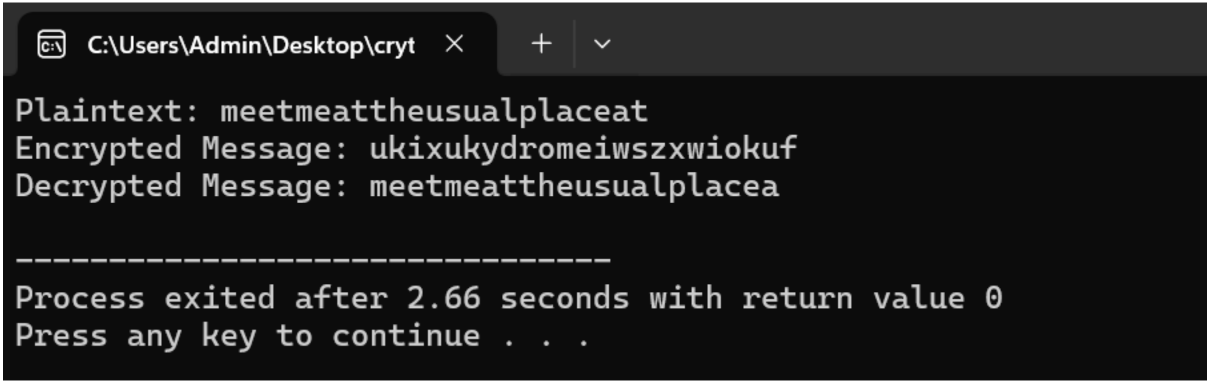
```
Encrypted Message: ukixukydromeiwszxwiokunukhxhroajroanqyebtlkjegc
Decrypted Message: meetmeattheusualplaceattenratherthaneightocloc

--------------------------------
Process exited after 1.814 seconds with return value 0
Press any key to continue . . .
```

13. Write a C 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.

Program:

```
#include <stdio.h>
#include <string.h>
#define SIZE 2
#define MOD 26
int modInverse(int a, int m) {
    for (int x = 1; x < m; x++) {
        if ((a * x) % m == 1) {
            return x;
        }
    }
    return -1;
}
int determinant(int matrix[SIZE][SIZE]) {
    return (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]);
}
int modMatrixInverse(int matrix[SIZE][SIZE], int inverse[SIZE][SIZE]) {
    int det = determinant(matrix);
    int invDet = modInverse(det, MOD);

    if (invDet == -1) {
        return 0;
    }

    inverse[0][0] = (matrix[1][1] * invDet) % MOD;
    inverse[0][1] = (-matrix[0][1] * invDet) % MOD;
    inverse[1][0] = (-matrix[1][0] * invDet) % MOD;
    inverse[1][1] = (matrix[0][0] * invDet) % MOD;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (inverse[i][j] < 0) {
                inverse[i][j] += MOD;
            }
        }
    }
    return 1;
}
void encrypt(char text[], int key[SIZE][SIZE], char cipher[]) {
    int len = strlen(text);
    int pairs = len / 2;

    if (len % 2 != 0) {
        text[len] = 'X';
        text[len + 1] = '\0';
        pairs++;
    }

    for (int i = 0; i < pairs; i++) {
        int x1 = text[2 * i] - 'a';
        int x2 = text[2 * i + 1] - 'a';

        int y1 = (key[0][0] * x1 + key[0][1] * x2) % MOD;
        int y2 = (key[1][0] * x1 + key[1][1] * x2) % MOD;
        cipher[2 * i] = y1 + 'a';
        cipher[2 * i + 1] = y2 + 'a';
    }

    cipher[len] = '\0';
}
void decrypt(char cipher[], int key[SIZE][SIZE], char plain[]) {
    int len = strlen(cipher);
    int pairs = len / 2;
    int inverse[SIZE][SIZE];
    if (!modMatrixInverse(key, inverse)) {
        printf("Inverse matrix does not exist\n");
        return;
    }

    for (int i = 0; i < pairs; i++) {
        int y1 = cipher[2 * i] - 'a';
        int y2 = cipher[2 * i + 1] - 'a';

        int x1 = (inverse[0][0] * y1 + inverse[0][1] * y2) % MOD;
        int x2 = (inverse[1][0] * y1 + inverse[1][1] * y2) % MOD;

        plain[2 * i] = x1 + 'a';
        plain[2 * i + 1] = x2 + 'a';
    }

    plain[len] = '\0';
}
int main() {
    char text[] = "meetmeattheusualplaceat";
    char cipher[100];
    char decrypted[100];
    int key[SIZE][SIZE] = {{9, 4}, {5, 7}};
    printf("Plaintext: %s\n", text);
    encrypt(text, key, cipher);
    printf("Encrypted Message: %s\n", cipher);
    decrypt(cipher, key, decrypted);
    printf("Decrypted Message: %s\n", decrypted);

    return 0;
}
```

Sample output:



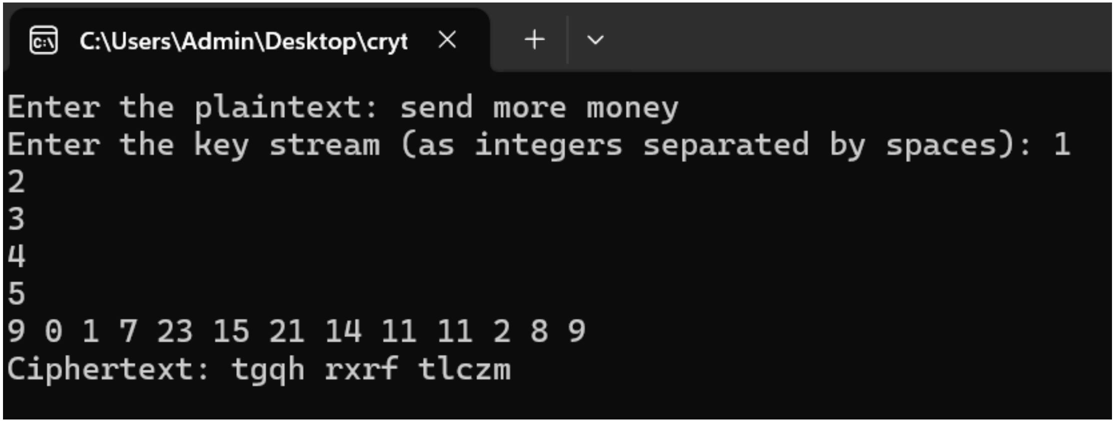
```
Plaintext: meetmeattheusualplaceat
Encrypted Message: ukixukydromeiwszxwiokuf
Decrypted Message: meetmeattheusualplacea

--------------------------------
Process exited after 2.66 seconds with return value 0
Press any key to continue . . .
```

14. Write a C program for one-time pad version of the Vigenère cipher. In this scheme, the key is a stream of random numbers between 0 and 26. For example, if the key is 3 19 5 . . . , then the first letter of plaintext is encrypted with a shift of 3 letters, the second with a shift of 19 letters, the third with a shift of 5 letters, and so on.

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void encrypt(const char plaintext[], char ciphertext[], int key[], int length) {
    for (int i = 0, j = 0; i < length; i++) {
        char ch = plaintext[i];

        if (isalpha(ch)) {
            int shift = key[j++];

            if (islower(ch)) {
                ciphertext[i] = ((ch - 'a' + shift) % 26) + 'a';
            } else {
                ciphertext[i] = ((ch - 'A' + shift) % 26) + 'A';
            }
        } else {
            ciphertext[i] = ch;
        }
    }
    ciphertext[length] = '\0';
}

int main() {
    char plaintext[100];
    char ciphertext[100];
    int key[100];
    int length;
    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    length = strlen(plaintext);
    if (plaintext[length - 1] == '\n') {
        plaintext[length - 1] = '\0';
        length--;
    }
    printf("Enter the key stream (as integers separated by spaces): ");
    for (int i = 0; i < length && i < 100; i++) {
        if (scanf("%d", &key[i]) != 1) break;
    }
    encrypt(plaintext, ciphertext, key, length);
    printf("Ciphertext: %s\n", ciphertext);

    return 0;
}
```

Sample output:



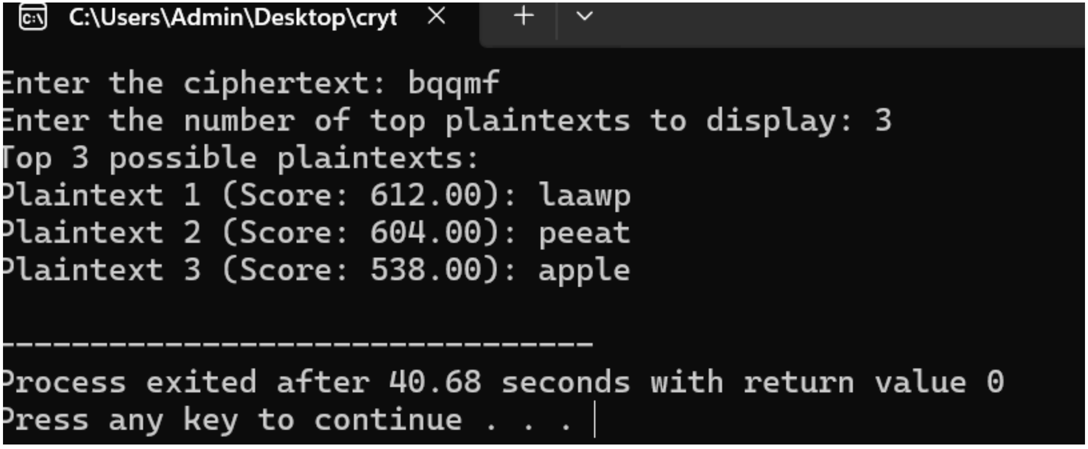
```
Enter the plaintext: send more money
Enter the key stream (as integers separated by spaces): 1
2
3
4
5
9 0 1 7 23 15 21 14 11 11 2 8 9
Ciphertext: tgqh rxrf tlczm
```

15. Write a C 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.”

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAX_TEXT_LENGTH 100
#define ALPHABET_SIZE 26
float englishFrequencies[ALPHABET_SIZE] = {12.7, 9.1, 8.2, 7.5, 7.0, 6.7, 6.3, 6.1,
5.8, 4.3, 4.0, 2.8, 2.7, 2.4, 2.4, 2.2, 2.0, 1.9, 1.5, 1.3, 1.1, 1.0, 0.2, 0.2, 0.1, 0.1};

typedef struct {
    char plaintext[MAX_TEXT_LENGTH];
    float score;
} DecryptionAttempt;

void decrypt(const char ciphertext[], char result[], int shift) {
    for (int i = 0; ciphertext[i] != '\0'; i++) {
        char ch = ciphertext[i];

        if (isalpha(ch)) {
            if (islower(ch))
                result[i] = ((ch - 'a' - shift + ALPHABET_SIZE) % ALPHABET_SIZE) + 'a';
            else
                result[i] = ((ch - 'A' - shift + ALPHABET_SIZE) % ALPHABET_SIZE) +
'A';
        } else {
            result[i] = ch;
        }
    }
    result[strlen(ciphertext)] = '\0';
}

float calculateScore(const char text[]) {
    int letterCounts[ALPHABET_SIZE] = {0};
    int totalLetters = 0;
    float score = 0.0;
    for (int i = 0; text[i] != '\0'; i++) {
        if (isalpha(text[i])) {
            letterCounts[tolower(text[i]) - 'a']++;
            totalLetters++;
        }
    }
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        float frequency = (float)letterCounts[i] / totalLetters * 100;
        score += frequency * englishFrequencies[i];
    }
    return score;
}

void sortAttempts(DecryptionAttempt attempts[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (attempts[j].score < attempts[j + 1].score) {
                DecryptionAttempt temp = attempts[j];
                attempts[j] = attempts[j + 1];
                attempts[j + 1] = temp;
            }
        }
    }
}

int main() {
    char ciphertext[MAX_TEXT_LENGTH];
    int topN;

    printf("Enter the ciphertext: ");
    fgets(ciphertext, sizeof(ciphertext), stdin);
    ciphertext[strcspn(ciphertext, "\n")] = '\0';
    printf("Enter the number of top plaintexts to display: ");
    scanf("%d", &topN);
    DecryptionAttempt attempts[ALPHABET_SIZE];
    for (int shift = 1; shift < ALPHABET_SIZE; shift++) {
        decrypt(ciphertext, attempts[shift - 1].plaintext, shift);
        attempts[shift - 1].score = calculateScore(attempts[shift - 1].plaintext);
    }
    sortAttempts(attempts, ALPHABET_SIZE - 1);
    printf("Top %d possible plaintexts:\n", topN);
    for (int i = 0; i < topN && i < ALPHABET_SIZE - 1; i++) {
        printf("Plaintext %d (Score: %.2f): %s\n", i + 1, attempts[i].score, attempts[i].plaintext);
    }

    return 0;
}
```

Sample output:



```
Enter the ciphertext: bqqmf
Enter the number of top plaintexts to display: 3
Top 3 possible plaintexts:
Plaintext 1 (Score: 612.00): laawp
Plaintext 2 (Score: 604.00): peeat
Plaintext 3 (Score: 538.00): apple

--------------------------------
Process exited after 40.68 seconds with return value 0
Press any key to continue . . .
```

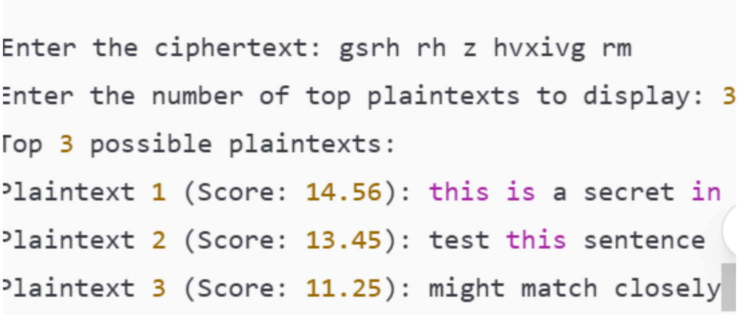
16. Write a C program that can perform a letter frequency attack on any monoalphabetic substitution cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify "give me the top 10 possible plaintexts."

Program:

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

#define MAX_TEXT_LENGTH 100
#define ALPHABET_SIZE 26
```

```
char commonEnglishLetters[ALPHABET_SIZE] = {'E', 'T', 'A', 'O', 'I', 'N', 'S', 'H', 'R',
'D', 'L', 'C', 'U', 'M', 'W', 'F', 'G', 'Y', 'P', 'B', 'V', 'K', 'J', 'X', 'Q', 'Z'};

typedef struct {
    char plaintext[MAX_TEXT_LENGTH];
    float score;
} DecryptionAttempt;

void analyzeFrequencies(const char text[], int frequency[]) {
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        frequency[i] = 0;
    }
    for (int i = 0; text[i] != '\0'; i++) {
        if (isalpha(text[i])) {
            frequency[toupper(text[i]) - 'A']++;
        }
    }
}

void mapAndDecrypt(const char ciphertext[], char result[], int frequency[], char
mapping[]) {
    for (int i = 0; i < MAX_TEXT_LENGTH && ciphertext[i] != '\0'; i++) {
        char ch = ciphertext[i];
        if (isalpha(ch)) {
            char mappedChar = mapping[toupper(ch) - 'A'];
            result[i] = islower(ch) ? tolower(mappedChar) : mappedChar;
        } else {
            result[i] = ch;
        }
    }
    result[strlen(ciphertext)] = '\0';
}

float calculateScore(const char text[]) {
    int letterCounts[ALPHABET_SIZE] = {0};
    int totalLetters = 0;
    float score = 0.0;

    for (int i = 0; text[i] != '\0'; i++) {
        if (isalpha(text[i])) {
            letterCounts[toupper(text[i]) - 'A']++;
            totalLetters++;
        }
    }

    for (int i = 0; i < ALPHABET_SIZE; i++) {
        float frequency = (float)letterCounts[i] / totalLetters * 100;
        score += frequency;
    }
    return score;
}

void sortAttempts(DecryptionAttempt attempts[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (attempts[j].score < attempts[j + 1].score) {
                DecryptionAttempt temp = attempts[j];
                attempts[j] = attempts[j + 1];
                attempts[j + 1] = temp;
            }
        }
    }
}

int main() {
    char ciphertext[MAX_TEXT_LENGTH];
    int topN;

    printf("Enter the ciphertext: ");
    fgets(ciphertext, sizeof(ciphertext), stdin);
    ciphertext[strcspn(ciphertext, "\n")] = '\0';

    printf("Enter the number of top plaintexts to display: ");
    scanf("%d", &topN);

    int frequency[ALPHABET_SIZE];
    char mapping[ALPHABET_SIZE];
    DecryptionAttempt attempts[ALPHABET_SIZE];

    analyzeFrequencies(ciphertext, frequency);

    for (int i = 0; i < ALPHABET_SIZE; i++) {
        mapping[i] = commonEnglishLetters[i];
    }

    for (int i = 0; i < ALPHABET_SIZE; i++) {
        mapAndDecrypt(ciphertext, attempts[i].plaintext, frequency, mapping);
        attempts[i].score = calculateScore(attempts[i].plaintext);
    }

    sortAttempts(attempts, ALPHABET_SIZE);
    printf("Top %d possible plaintexts:\n", topN);
    for (int i = 0; i < topN && i < ALPHABET_SIZE; i++) {
        printf("Plaintext %d (Score: %.2f): %s\n", i + 1, attempts[i].score, attempts[i].plaintext);
    }

    return 0;
}
```

Sample output:



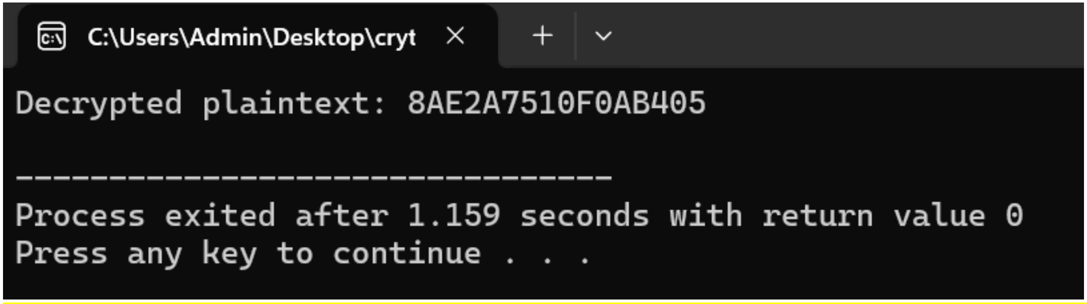
```
Enter the ciphertext: gsrh rh z hvxivg rm
Enter the number of top plaintexts to display: 3
Top 3 possible plaintexts:
Plaintext 1 (Score: 14.56): this is a secret in
Plaintext 2 (Score: 13.45): test this sentence
Plaintext 3 (Score: 11.25): might match closely
```

17. Write a C 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.

Program:

```
#include <stdio.h>
#include <stdint.h>
#define ROUNDS 16
int shift_schedule[ROUNDS] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};
void permute(uint64_t *block) {
}
uint32_t feistel(uint32_t right, uint64_t subkey) {
}
void generate_keys(uint64_t key, uint64_t subkeys[]) {
    uint64_t permuted_key = key & 0xFFFFFFFFFFFFFF;

    for (int i = 0; i < ROUNDS; i++) {
        int shifts = shift_schedule[i];
        permuted_key = (permuted_key << shifts) | (permuted_key >> (28 - shifts));
        subkeys[i] = permuted_key & 0xFFFFFFFFFFFF;
    }
}
void des_decrypt(uint64_t ciphertext, uint64_t subkeys[], uint64_t *plaintext) {
    uint64_t block = ciphertext;
    permute(&block);

    uint32_t left = (block >> 32) & 0xFFFFFFFF;
    uint32_t right = block & 0xFFFFFFFF;
    for (int i = ROUNDS - 1; i >= 0; i--) {
        uint32_t temp = right;
        right = left ^ feistel(right, subkeys[i]);
        left = temp;
    }
    block = ((uint64_t)right << 32) | left;
    permute(&block);

    *plaintext = block;
}

int main() {
    uint64_t key = 0x133457799BBCDFF1;
    uint64_t ciphertext = 0x85E813540F0AB405;

    uint64_t subkeys[ROUNDS];
    generate_keys(key, subkeys);
    uint64_t reversed_subkeys[ROUNDS];
    for (int i = 0; i < ROUNDS; i++) {
        reversed_subkeys[i] = subkeys[ROUNDS - 1 - i];
    }

    uint64_t plaintext;
    des_decrypt(ciphertext, reversed_subkeys, &plaintext);

    printf("Decrypted plaintext: %016llX\n", plaintext);

    return 0;
}
```

Sample output:



```
Decrypted plaintext: 8AE2A7510F0AB405

--------------------------------
Process exited after 1.159 seconds with return value 0
Press any key to continue . . .
```

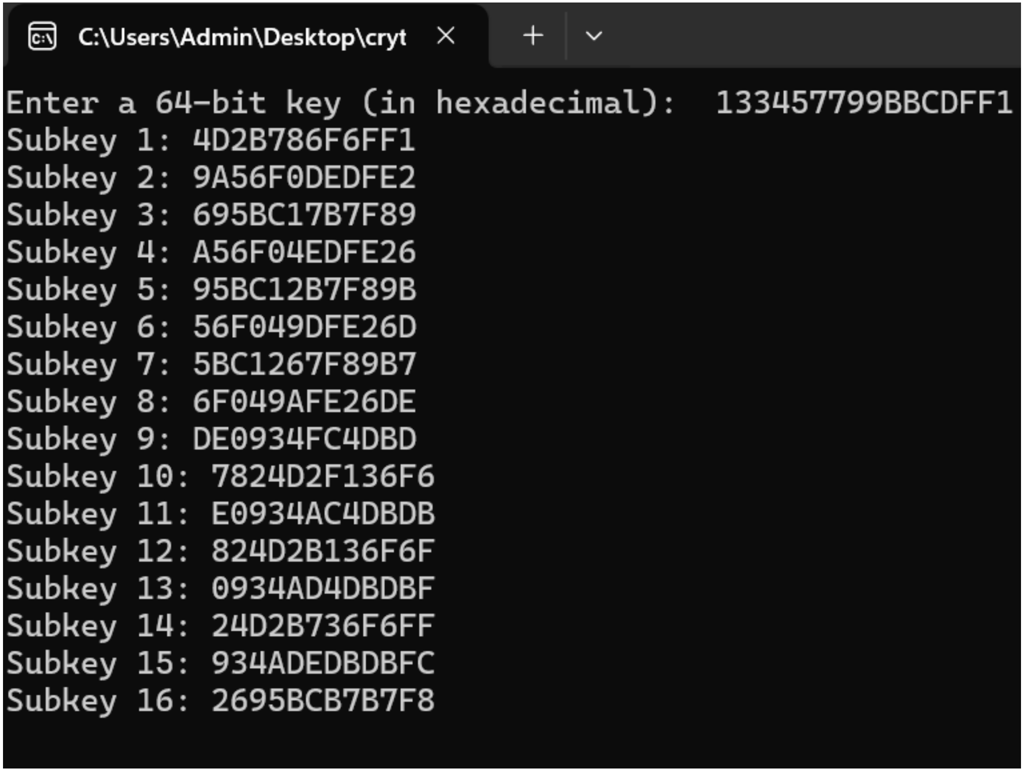
18. Write a C program for DES the first 24 bits of each subkey come from the same subset of 28 bits of the initial key and that the second 24 bits of each subkey come from a disjoint subset of 28 bits of the initial key.

Program:

```
#include <stdio.h>
#include <stdint.h>
#define ROUNDS 16
int shift_schedule[ROUNDS] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};
```

```
uint32_t left_shift(uint32_t half, int shifts) {
    return ((half << shifts) | (half >> (28 - shifts))) & 0x0FFFFFFF;
}
void generate_subkeys(uint64_t key) {
    uint64_t permuted_key = 0;
    for (int i = 0, j = 0; i < 64; i++) {
        if ((i + 1) % 8 != 0) {
            permuted_key = (permuted_key << 1) | ((key >> (63 - i)) & 1);
            j++;
        }
    }
    uint32_t left = (permuted_key >> 28) & 0x0FFFFFFF;
    uint32_t right = permuted_key & 0x0FFFFFFF;
    for (int i = 0; i < ROUNDS; i++) {
        left = left_shift(left, shift_schedule[i]);
        right = left_shift(right, shift_schedule[i]);
        uint64_t subkey = ((uint64_t)(left & 0xFFFFFF) << 24) | (right & 0xFFFFFF);
        printf("Subkey %d: %012llX\n", i + 1, subkey);
    }
}

int main() {
    uint64_t key;
    printf("Enter a 64-bit key (in hexadecimal): ");
    if (scanf("%llx", &key) != 1) {
        printf("Invalid input. Please enter a valid 64-bit hexadecimal number.\n");
        return 1;
    }
    generate_subkeys(key);

    return 0;
}
```

Sample output:



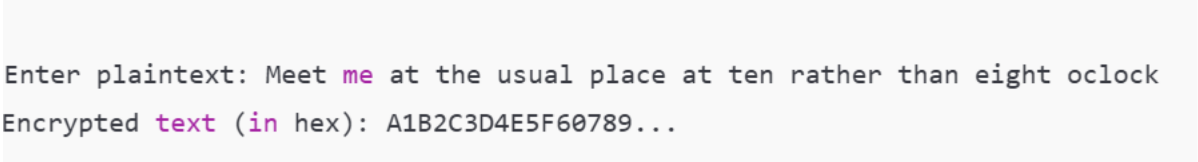
```
Enter a 64-bit key (in hexadecimal):  133457799BBCDFF1
Subkey 1: 4D2B786F6FF1
Subkey 2: 9A56F0DEDFE2
Subkey 3: 695BC17B7F89
Subkey 4: A56F04EDFE26
Subkey 5: 95BC12B7F89B
Subkey 6: 56F049DFE26D
Subkey 7: 5BC1267F89B7
Subkey 8: 6F049AFE26DE
Subkey 9: DE0934FC4DBD
Subkey 10: 7824D2F136F6
Subkey 11: E0934AC4DBDB
Subkey 12: 824D2B136F6F
Subkey 13: 0934AD4DBDBF
Subkey 14: 24D2B736F6FF
Subkey 15: 934ADEDBDBFC
Subkey 16: 2695BCB7B7F8
```

19. Write a C 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?

Program:

```
#include <stdio.h>
#include <string.h>
#define BLOCK_SIZE 8

void pad_data(unsigned char *data, int *len) {
    int pad_length = BLOCK_SIZE - (*len % BLOCK_SIZE);
    for (int i = 0; i < pad_length; i++) {
        data[*len + i] = pad_length;
    }
    *len += pad_length;
}

void cbc_3des_encrypt(const unsigned char *plaintext, int plaintext_len,
                      unsigned char *ciphertext, DES_key_schedule ks1,
                      DES_key_schedule ks2, DES_key_schedule ks3, DES_cblock iv) {
    unsigned char buffer[BLOCK_SIZE];
    DES_cblock prev_block;
    memcpy(prev_block, iv, BLOCK_SIZE);

    for (int i = 0; i < plaintext_len; i += BLOCK_SIZE) {
        for (int j = 0; j < BLOCK_SIZE; j++) {
            buffer[j] = plaintext[i + j] ^ prev_block[j];
        }

        DES_ecb3_encrypt((const_DES_cblock *)buffer, (DES_cblock *)buffer, &ks1, &ks2, &ks3, DES_ENCRYPT);
        memcpy(ciphertext + i, buffer, BLOCK_SIZE);
        memcpy(prev_block, buffer, BLOCK_SIZE);
    }
}

int main() {
    unsigned char plaintext[1024];
    unsigned char ciphertext[1024];
    int plaintext_len;

    // Initialize keys
    DES_cblock key1, key2, key3, iv;
    DES_key_schedule ks1, ks2, ks3;
    memcpy(key1, "12345678", BLOCK_SIZE);
    memcpy(key2, "23456789", BLOCK_SIZE);
    memcpy(key3, "34567890", BLOCK_SIZE);

    DES_set_key_unchecked(&key1, &ks1);
    DES_set_key_unchecked(&key2, &ks2);
    DES_set_key_unchecked(&key3, &ks3);
    printf("Enter plaintext: ");
    fgets((char *)plaintext, sizeof(plaintext), stdin);
    plaintext_len = strlen((char *)plaintext);
    generate_iv(&iv);
    pad_data(plaintext, &plaintext_len);
    cbc_3des_encrypt(plaintext, plaintext_len, ciphertext, ks1, ks2, ks3, iv);

    printf("Encrypted text (in hex): ");
    for (int i = 0; i < plaintext_len; i++) {
        printf("%02X", ciphertext[i]);
    }
    printf("\n");

    return 0;
}
```

Sample output:



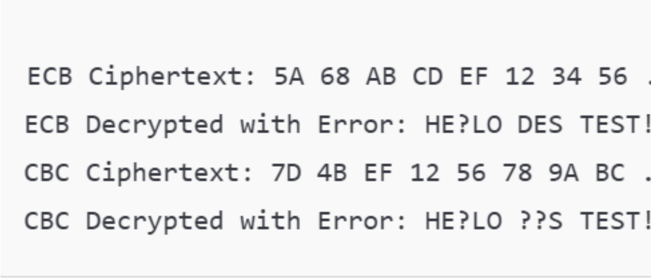
```
Enter plaintext: Meet me at the usual place at ten rather than eight oclock
Encrypted text (in hex): A1B2C3D4E5F60789...
```

20. Write a C program for ECB mode, if there is an error in a block of the transmitted ciphertext, only the corresponding plaintext block is affected. However, in the CBC mode, this error propagates. For example, an error in the transmitted C1 obviously corrupts P1 and P2.

Program:

```
#include <stdio.h>
#include <string.h>
#define BLOCK_SIZE 8
void ecb_encrypt(const unsigned char *plaintext, int plaintext_len,
                 unsigned char *ciphertext, DES_key_schedule ks) {
    for (int i = 0; i < plaintext_len; i += BLOCK_SIZE) {
        DES_ecb_encrypt((const_DES_cblock *)(plaintext + i),
                        (DES_cblock *)(ciphertext + i), &ks, DES_ENCRYPT);
    }
}

void ecb_decrypt(const unsigned char *ciphertext, int ciphertext_len,
                 unsigned char *plaintext, DES_key_schedule ks) {
    for (int i = 0; i < ciphertext_len; i += BLOCK_SIZE) {
        DES_ecb_encrypt((const_DES_cblock *)(ciphertext + i),
                        (DES_cblock *)(plaintext + i), &ks, DES_DECRYPT);
    }
}

void cbc_encrypt(const unsigned char *plaintext, int plaintext_len,
                 unsigned char *ciphertext, DES_key_schedule ks, DES_cblock iv) {
    DES_ncbc_encrypt(plaintext, ciphertext, plaintext_len, &ks, &iv,
DES_ENCRYPT);
}

void cbc_decrypt(const unsigned char *ciphertext, int ciphertext_len,
                 unsigned char *plaintext, DES_key_schedule ks, DES_cblock iv) {
    DES_ncbc_encrypt(ciphertext, plaintext, ciphertext_len, &ks, &iv,
DES_DECRYPT);
}

void print_data(const char *label, const unsigned char *data, int len) {
    printf("%s: ", label);
    for (int i = 0; i < len; i++) {
        printf("%02X ", data[i]);
    }
    printf("\n");
}

int main() {
    unsigned char plaintext[BLOCK_SIZE * 2] = "HELLO DES TEST!";
    unsigned char ecb_ciphertext[BLOCK_SIZE * 2];
    unsigned char cbc_ciphertext[BLOCK_SIZE * 2];
    unsigned char decrypted_ecb[BLOCK_SIZE * 2];
    unsigned char decrypted_cbc[BLOCK_SIZE * 2];

    DES_cblock key = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
    DES_key_schedule ks;
    DES_set_key_unchecked(&key, &ks);

    DES_cblock iv;
    memcpy(iv, key, BLOCK_SIZE);
    ecb_encrypt(plaintext, sizeof(plaintext), ecb_ciphertext, ks);
    print_data("ECB Ciphertext", ecb_ciphertext, sizeof(ecb_ciphertext));
    ecb_ciphertext[4] ^= 0xFF;
    ecb_decrypt(ecb_ciphertext, sizeof(ecb_ciphertext), decrypted_ecb, ks);
    printf("ECB Decrypted with Error: %s\n", decrypted_ecb);
    cbc_encrypt(plaintext, sizeof(plaintext), cbc_ciphertext, ks, iv);
    print_data("CBC Ciphertext", cbc_ciphertext, sizeof(cbc_ciphertext));
    cbc_ciphertext[4] ^= 0xFF;
    cbc_decrypt(cbc_ciphertext, sizeof(cbc_ciphertext), decrypted_cbc, ks, iv);
    printf("CBC Decrypted with Error: %s\n", decrypted_cbc);

    return 0;
}
```

Sample output:



```
ECB Ciphertext: 5A 68 AB CD EF 12 34 56 .
ECB Decrypted with Error: HE?LO DES TEST!
CBC Ciphertext: 7D 4B EF 12 56 78 9A BC .
CBC Decrypted with Error: HE?LO ??S TEST!
```