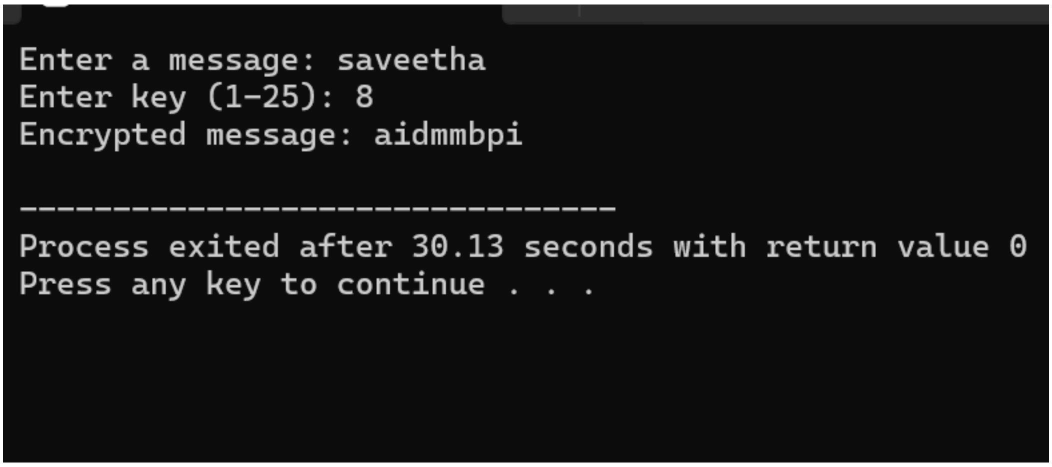
1.Code:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAX_LEN 1000
void caesarCipher(char *text, int key) {
    for (int i = 0; text[i] != '\0'; i++) {
        char c = text[i];

        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            text[i] = (c - base + key) % 26 + base;
        }
    }
}
int main() {
    char text[MAX_LEN];
    int key;
    printf("Enter a message: ");
    fgets(text, sizeof(text), stdin);
    text[strcspn(text, "\n")] = '\0';
    printf("Enter key (1-25): ");
    scanf("%d", &key);
    if (key < 1 || key > 25) {
        printf("Invalid key. Must be between 1 and 25.\n");
        return 1;
    }
    caesarCipher(text, key);

    printf("Encrypted message: %s\n", text);

    return 0;

}
```

Output:



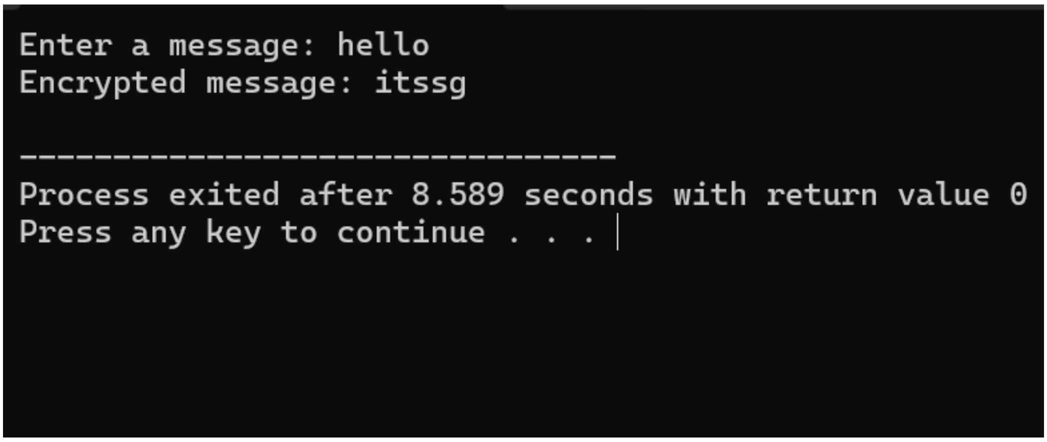
```
Enter a message: saveetha
Enter key (1-25): 8
Encrypted message: aidmmbpi

--------------------------------
Process exited after 30.13 seconds with return value 0
Press any key to continue . . .
```

2.Code:

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX_LEN 1000

void monoalphabeticEncrypt(char *plaintext, const char *key) {

    for (int i = 0; plaintext[i] != '\0'; i++) {

        if (isupper(plaintext[i])) {

            plaintext[i] = toupper(key[plaintext[i] - 'A']);

        } else if (islower(plaintext[i])) {

            plaintext[i] = tolower(key[plaintext[i] - 'a']);

        }

    }

}
```

```
int main() {
    char plaintext[MAX_LEN];
    char key[27] = "QWERTYUIOPASDFGHJKLZXCVBNM";
    printf("Enter a message: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    if (strlen(key) != 26) {
        printf("Invalid key. Must be 26 letters.\n");
        return 1;
    }
    monoalphabeticEncrypt(plaintext, key);
    printf("Encrypted message: %s\n", plaintext);
    return 0;
}
```

Output:



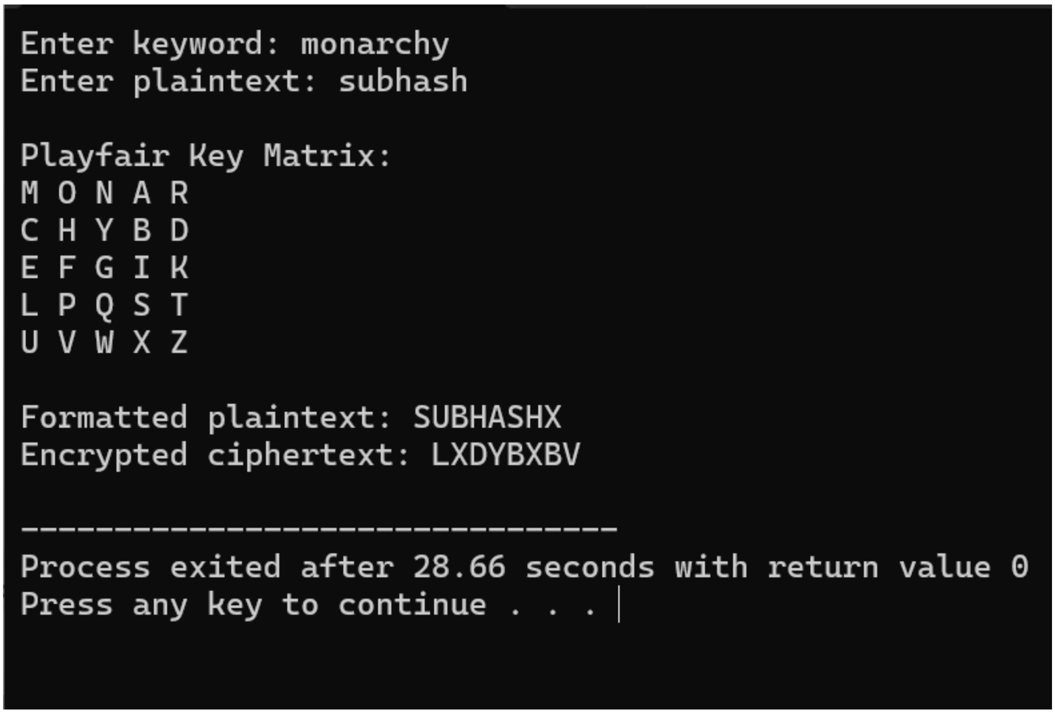
```
Enter a message: hello
Encrypted message: itssg

--------------------------------
Process exited after 8.589 seconds with return value 0
Press any key to continue . . .
```

3.Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 5
```

```
char matrix[SIZE][SIZE];
void prepareKeyMatrix(char *key) {
    int used[26] = {0};
    int x = 0, y = 0;
    char c;
    for (int i = 0; key[i]; i++) {
        c = toupper(key[i]);
        if (c == 'J') c = 'I';
        if (isalpha(c) && !used[c - 'A']) {
            matrix[x][y++] = c;
            used[c - 'A'] = 1;
            if (y == SIZE) {
                y = 0;
                x++;
            }
        }
    }
    for (c = 'A'; c <= 'Z'; c++) {
        if (c == 'J') continue;
        if (!used[c - 'A']) {
            matrix[x][y++] = c;
            used[c - 'A'] = 1;
            if (y == SIZE) {
                y = 0;
                x++;
            }
        }
    }
}
void printMatrix() {
    printf("\nPlayfair Key Matrix:\n");
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++)
            printf("%c ", matrix[i][j]);
        printf("\n");
    }
}
void findPosition(char ch, int *row, int *col) {
    if (ch == 'J') ch = 'I';
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (matrix[i][j] == ch) {
                *row = i;
                *col = j;
                return;
            }
        }
    }
}
void formatPlaintext(const char *input, char *output) {
    char temp[100];
    int k = 0;
    for (int i = 0; input[i]; i++) {
        if (isalpha(input[i])) {
            temp[k++] = toupper(input[i]) == 'J' ? 'I' : toupper(input[i]);
        }
    }
    int j = 0;
    for (int i = 0; i < k; i++) {
        output[j++] = temp[i];
        if (i + 1 < k && temp[i] == temp[i + 1]) {
            output[j++] = 'X';
        } else if (i + 1 < k) {
            output[j++] = temp[++i];
        }
    }
    if (j % 2 != 0) output[j++] = 'X';
    output[j] = '\0';
}
void encrypt(const char *plaintext, char *ciphertext) {
    int r1, c1, r2, c2;
    for (int i = 0; plaintext[i] && plaintext[i + 1]; i += 2) {
        findPosition(plaintext[i], &r1, &c1);
        findPosition(plaintext[i + 1], &r2, &c2);
        if (r1 == r2) {
            ciphertext[i] = matrix[r1][(c1 + 1) % SIZE];
            ciphertext[i + 1] = matrix[r2][(c2 + 1) % SIZE];
        } else if (c1 == c2) {
            ciphertext[i] = matrix[(r1 + 1) % SIZE][c1];
            ciphertext[i + 1] = matrix[(r2 + 1) % SIZE][c2];
        } else {
            ciphertext[i] = matrix[r1][c2];
            ciphertext[i + 1] = matrix[r2][c1];
        }
    }
    ciphertext[strlen(plaintext)] = '\0';
}
int main() {
    char key[100], plaintext[100], formatted[100], ciphertext[100];
    printf("Enter keyword: ");
    scanf("%s", key);
    printf("Enter plaintext: ");
    scanf(" %[^\n]", plaintext);
    prepareKeyMatrix(key);
    printMatrix();
    formatPlaintext(plaintext, formatted);
    printf("\nFormatted plaintext: %s\n", formatted);
    encrypt(formatted, ciphertext);
    printf("Encrypted ciphertext: %s\n", ciphertext);
    return 0;
}
```

Output:

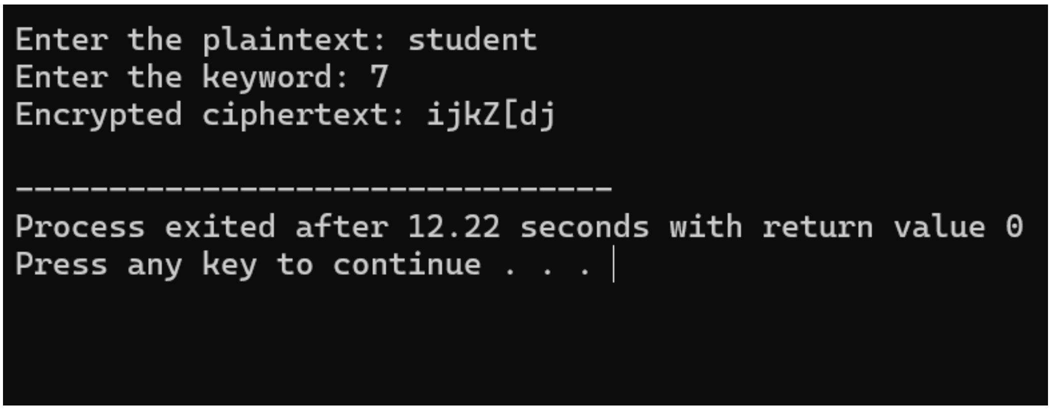


```
Enter keyword: monarchy
Enter plaintext: subhash

Playfair Key Matrix:
M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z

Formatted plaintext: SUBHASHX
Encrypted ciphertext: LXDYBXBV

--------------------------------
Process exited after 28.66 seconds with return value 0
Press any key to continue . . .
```

4.Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int charToShift(char c) {
    return toupper(c) - 'A';
}
void polyalphabeticEncrypt(char *plaintext, char *key, char *ciphertext) {
    int textLen = strlen(plaintext);
    int keyLen = strlen(key);
    int j = 0;
    for (int i = 0; i < textLen; i++) {
        char pt = plaintext[i];
        if (isalpha(pt)) {
            char k = toupper(key[j % keyLen]);
            int shift = charToShift(k);
            if (isupper(pt)) {
                ciphertext[i] = ((pt - 'A' + shift) % 26) + 'A';
            } else {
                ciphertext[i] = ((pt - 'a' + shift) % 26) + 'a';
            }
            j++;
        } else {
            ciphertext[i] = pt;
        }
    }
    ciphertext[textLen] = '\0';
}
int main() {
    char plaintext[1000], key[100], ciphertext[1000];
    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    printf("Enter the keyword: ");
    scanf("%s", key);
    polyalphabeticEncrypt(plaintext, key, ciphertext);
    printf("Encrypted ciphertext: %s\n", ciphertext);
    return 0;
}
```

Output:

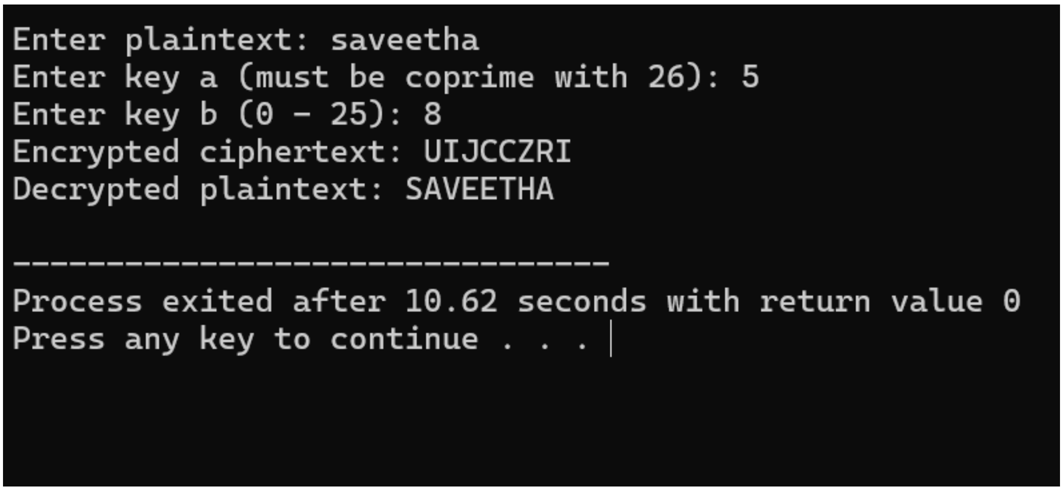


```
Enter the plaintext: student
Enter the keyword: 7
Encrypted ciphertext: ijkZ[dj

--------------------------------
Process exited after 12.22 seconds with return value 0
Press any key to continue . . .
```

5.Code:

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MOD 26

int modInverse(int a) {

    for (int i = 1; i < MOD; i++) {

        if ((a * i) % MOD == 1)

            return i;

    }

    return -1;

}

char affineEncryptChar(char ch, int a, int b) {

    if (isalpha(ch)) {

        ch = toupper(ch);

        return ((a * (ch - 'A') + b) % MOD) + 'A';

    }

    return ch;

}

char affineDecryptChar(char ch, int a, int b) {
    if (isalpha(ch)) {
        ch = toupper(ch);
        int a_inv = modInverse(a);
        if (a_inv == -1) return '?';
        int decrypted = (a_inv * ((ch - 'A') - b + MOD)) % MOD;
        return decrypted + 'A';
    }
    return ch;
}
void affineEncrypt(char *plaintext, char *ciphertext, int a, int b) {
    for (int i = 0; plaintext[i]; i++) {
        ciphertext[i] = affineEncryptChar(plaintext[i], a, b);
    }
    ciphertext[strlen(plaintext)] = '\0';
}
void affineDecrypt(char *ciphertext, char *plaintext, int a, int b) {
    for (int i = 0; ciphertext[i]; i++) {
        plaintext[i] = affineDecryptChar(ciphertext[i], a, b);
    }
    plaintext[strlen(ciphertext)] = '\0';
}
int main() {
    char plaintext[100], ciphertext[100], decrypted[100];
    int a, b;
    printf("Enter plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    printf("Enter key a (must be coprime with 26): ");
    scanf("%d", &a);
    printf("Enter key b (0 - 25): ");
    scanf("%d", &b);
    if (modInverse(a) == -1) {
        printf("Invalid key 'a'. It must be coprime with 26.\n");
        return 1;
    }
    affineEncrypt(plaintext, ciphertext, a, b);
    printf("Encrypted ciphertext: %s\n", ciphertext);
    affineDecrypt(ciphertext, decrypted, a, b);
    printf("Decrypted plaintext: %s\n", decrypted);
    return 0;
}
```

Output:



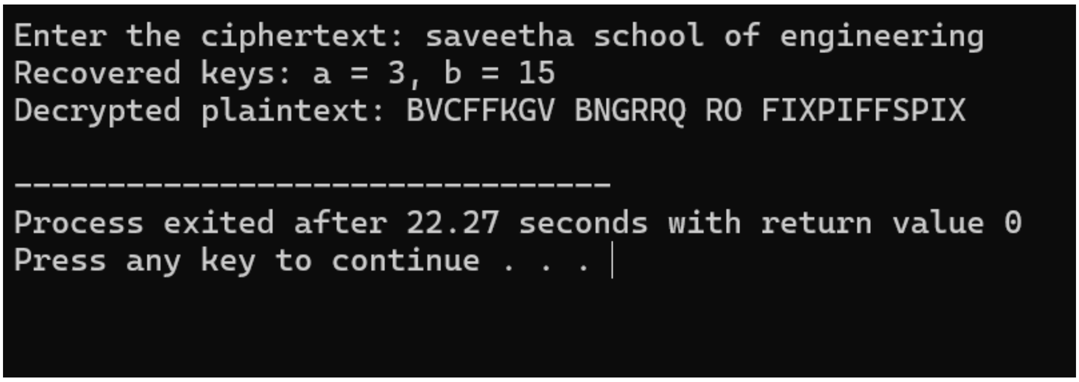
```
Enter plaintext: saveetha
Enter key a (must be coprime with 26): 5
Enter key b (0 - 25): 8
Encrypted ciphertext: UIJCCZRI
Decrypted plaintext: SAVEETHA

--------------------------------
Process exited after 10.62 seconds with return value 0
Press any key to continue . . .
```

6.Code:

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

```
#define MOD 26
int modInverse(int a) {
    for (int i = 1; i < MOD; i++) {
        if ((a * i) % MOD == 1) return i;
    }
    return -1;
}
char affineDecryptChar(char c, int a, int b) {
    if (!isalpha(c)) return c;
    c = toupper(c);
    int a_inv = modInverse(a);
    if (a_inv == -1) return '?';

    int x = ((a_inv * ((c - 'A') - b + MOD)) % MOD);
    return x + 'A';
}
int solveAffineKeys(int p1, int c1, int p2, int c2, int *a, int *b) {
    int delta_p = (p1 - p2 + MOD) % MOD;
    int delta_c = (c1 - c2 + MOD) % MOD;
    int inv = modInverse(delta_p);
    if (inv == -1) return 0;
    *a = (delta_c * inv) % MOD;
    *b = (c1 - (*a * p1) + MOD * MOD) % MOD;
    return 1;
}
void decryptCiphertext(const char *ciphertext, char *plaintext, int a, int b) {
    for (int i = 0; ciphertext[i]; i++) {
        plaintext[i] = affineDecryptChar(ciphertext[i], a, b);
    }
    plaintext[strlen(ciphertext)] = '\0';
}
int main() {
    char ciphertext[1000], plaintext[1000];
    int a, b;
    char c1 = 'B';
    char p1 = 'E';
    char c2 = 'U';
    char p2 = 'T';
    printf("Enter the ciphertext: ");
    fgets(ciphertext, sizeof(ciphertext), stdin);
    ciphertext[strcspn(ciphertext, "\n")] = '\0';
    int success = solveAffineKeys(p1 - 'A', c1 - 'A', p2 - 'A', c2 - 'A', &a, &b);
    if (!success || modInverse(a) == -1) {
        printf("Failed to break the cipher using current assumptions.\n");
        return 1;
    }
    printf("Recovered keys: a = %d, b = %d\n", a, b);
    decryptCiphertext(ciphertext, plaintext, a, b);
    printf("Decrypted plaintext: %s\n", plaintext);
    return 0;
}
```

Output:



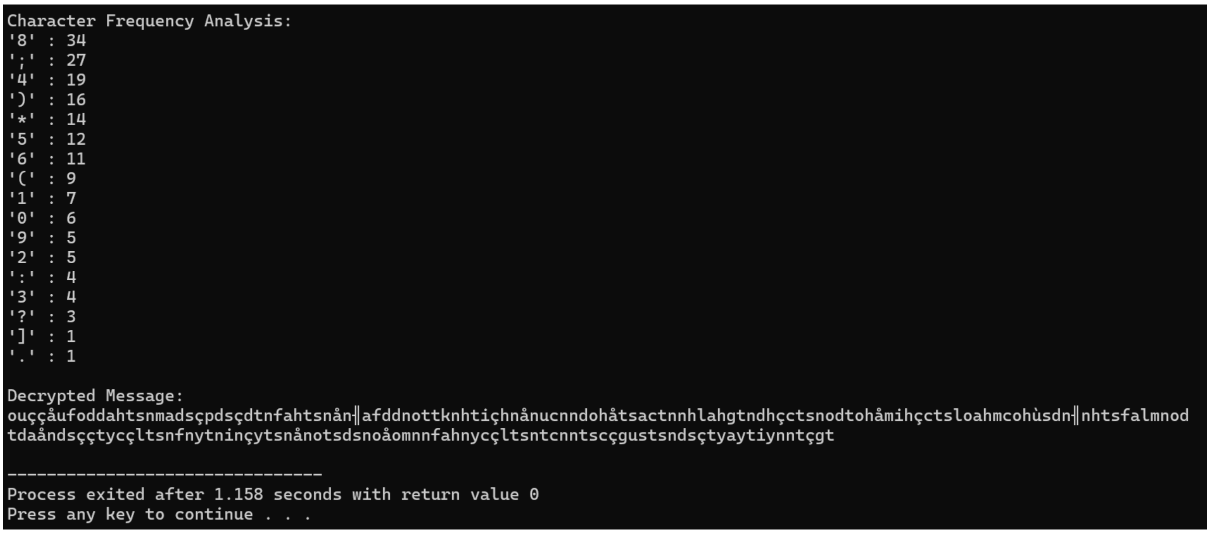
```
Enter the ciphertext: saveetha school of engineering
Recovered keys: a = 3, b = 15
Decrypted plaintext: BVCFFKGV BNGRRQ RO FIXPIFFSPIX

--------------------------------
Process exited after 22.27 seconds with return value 0
Press any key to continue . . .
```

7.Code:

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX_LEN 1000

void countFrequencies(char *text, int *freq) {

   for (int i = 0; text[i]; i++) {

      freq[(unsigned char)text[i]]++;

   }

}

void printSortedFrequencies(int *freq) {

   int sorted[256];

   for (int i = 0; i < 256; i++) sorted[i] = i;

   for (int i = 0; i < 255; i++) {

      for (int j = i + 1; j < 256; j++) {

         if (freq[sorted[i]] < freq[sorted[j]]) {

            int temp = sorted[i];

            sorted[i] = sorted[j];

            sorted[j] = temp;

         }

      }
```

```
    }
    printf("Character Frequency Analysis:\n");
    for (int i = 0; i < 256; i++) {
        if (freq[sorted[i]] > 0 && isprint(sorted[i]))
            printf("'%c' : %d\n", sorted[i], freq[sorted[i]]);
    }
}
void decryptMessage(char *cipher, char map[256]) {
    printf("\nDecrypted Message:\n");
    for (int i = 0; cipher[i]; i++) {
        char ch = cipher[i];
        if (map[(unsigned char)ch] != 0)
            putchar(map[(unsigned char)ch]);
        else
            putchar(ch);
    }
    printf("\n");
}
int main() {
    char ciphertext[MAX_LEN] =
        "53‡‡†305))6*;4826)4‡.)4‡);806*;48†8¶60))85;;]8*;:‡*8†83"
        "(88)5*†;46(;88*96*?;8)*‡(;485);5*†2:*‡(;4956*2(5*—4)8¶8*;"
        "4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806*81"
        "(‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;";
    int freq[256] = {0};
    countFrequencies(ciphertext, freq);
    printSortedFrequencies(freq);
    char map[256] = {0};
    map['‡'] = 'e';
    map[';'] = 't';
    map['*'] = 'h';
    map['5'] = 'o';
    map['8'] = 'n';
    map['4'] = 's';
    map['†'] = 'r';
    map['6'] = 'a';
    map[')'] = 'd';
    map['3'] = 'u';
    map['0'] = 'f';
    map['9'] = 'l';
    map['2'] = 'm';
    map[':'] = 'i';
    map['1'] = 'y';
    map['('] = 'c';
    map['?'] = 'g';
    map['.'] = 'p';
    map['['] = 'b';
    map[']'] = 'k';
    map['—'] = 'w';
    map['¶'] = 'v';
    decryptMessage(ciphertext, map);
    return 0;
}
```

Ouput:

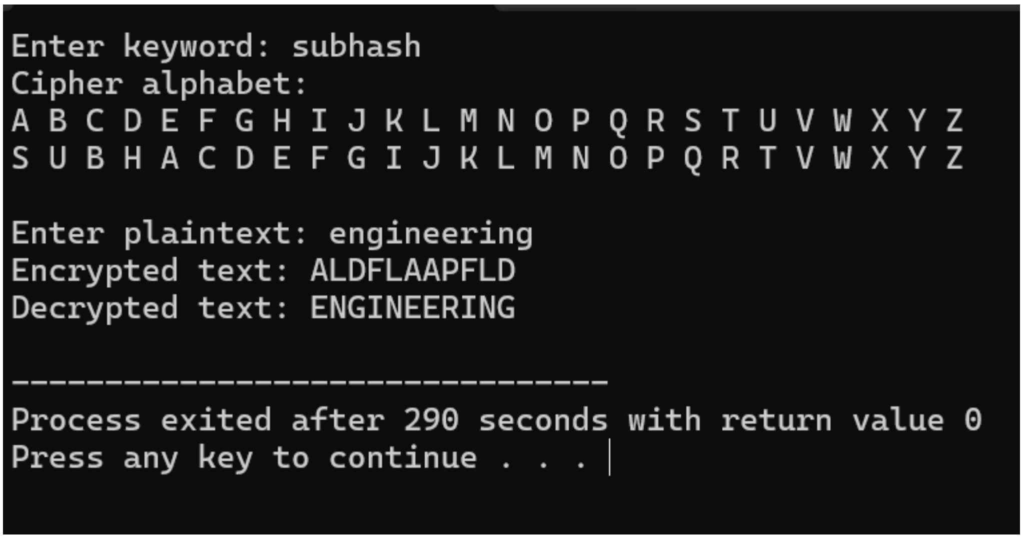


```
Character Frequency Analysis:
'8' : 34
';' : 27
'4' : 19
')' : 16
'*' : 14
'5' : 12
'6' : 11
'(' : 9
'1' : 7
'0' : 6
'9' : 5
'2' : 5
':' : 4
'3' : 4
'?' : 3
']' : 1
'.' : 1

Decrypted Message:
ouççåufoddahtsnmadsçpdsçdtnfahtsnån╢afddnottknhtiçhnånucnndohåtsactnnhlahgtndhçctsnodtohåmihçctsloahmcohùsdn╢nhtsfalmnod
tdaåndsççtycçltsnfnytninçytsnånotsdsnoåomnnfahnycçltsntcnntscçgustsndsçtyaytiynntçgt

--------------------------------
Process exited after 1.158 seconds with return value 0
Press any key to continue . . .
```

8.Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define ALPHABET_LEN 26
void generateCipherAlphabet(char *keyword, char *cipher) {
    int used[26] = {0};
    int i, j = 0;
    for (i = 0; keyword[i] != '\0'; i++) {
        char c = toupper(keyword[i]);
        if (isalpha(c) && !used[c - 'A']) {
            cipher[j++] = c;
            used[c - 'A'] = 1;
        }
    }
    for (i = 0; i < 26; i++) {
        if (!used[i]) {
            cipher[j++] = 'A' + i;
        }
    }
    cipher[j] = '\0';
}
void encrypt(const char *plain, char *cipherText, char *cipher) {
    for (int i = 0; plain[i] != '\0'; i++) {
        if (isalpha(plain[i])) {
            char c = toupper(plain[i]);
            cipherText[i] = cipher[c - 'A'];
        } else {
            cipherText[i] = plain[i];
        }
    }
    cipherText[strlen(plain)] = '\0';
}
void decrypt(const char *cipherText, char *plainText, char *cipher) {
    for (int i = 0; cipherText[i] != '\0'; i++) {
        if (isalpha(cipherText[i])) {
            char c = toupper(cipherText[i]);
            for (int j = 0; j < ALPHABET_LEN; j++) {
                if (cipher[j] == c) {
                    plainText[i] = 'A' + j;
                    break;
                }
            }
        } else {
            plainText[i] = cipherText[i];
        }
    }
    plainText[strlen(cipherText)] = '\0';
}
int main() {
    char keyword[100], cipher[27];
    char plainText[1024], cipherText[1024], decryptedText[1024];
    printf("Enter keyword: ");
    scanf("%s", keyword);
    generateCipherAlphabet(keyword, cipher);
    printf("Cipher alphabet:\n");
    for (int i = 0; i < ALPHABET_LEN; i++) {
        printf("%c ", 'A' + i);
    }
    printf("\n");
    for (int i = 0; i < ALPHABET_LEN; i++) {
        printf("%c ", cipher[i]);
    }
    printf("\n");
    printf("\nEnter plaintext: ");
    getchar();
    fgets(plainText, sizeof(plainText), stdin);
    plainText[strcspn(plainText, "\n")] = 0;
    encrypt(plainText, cipherText, cipher);
    printf("Encrypted text: %s\n", cipherText);
    decrypt(cipherText, decryptedText, cipher);
    printf("Decrypted text: %s\n", decryptedText);
    return 0;
}
```

Output:

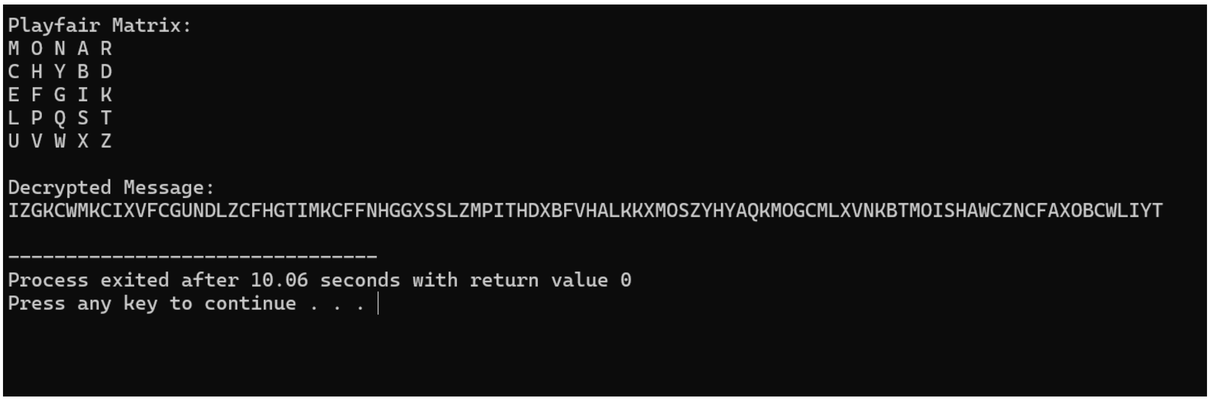


```
Enter keyword: subhash
Cipher alphabet:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
S U B H A C D E F G I J K L M N O P Q R T V W X Y Z

Enter plaintext: engineering
Encrypted text: ALDFLAAPFLD
Decrypted text: ENGINEERING

--------------------------------
Process exited after 290 seconds with return value 0
Press any key to continue . . .
```

9.Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 5
char matrix[SIZE][SIZE];
void generateMatrix(char *key) {
    int used[26] = {0};
    int x = 0, y = 0;
    used['J' - 'A'] = 1;
    for (int i = 0; key[i]; i++) {
        char c = toupper(key[i]);
        if (!isalpha(c)) continue;
        if (c == 'J') c = 'I';
        if (!used[c - 'A']) {
            matrix[x][y++] = c;
            used[c - 'A'] = 1;
            if (y == SIZE) {
                y = 0;
                x++;
            }
        }
    }
    for (char c = 'A'; c <= 'Z'; c++) {
        if (!used[c - 'A']) {
            matrix[x][y++] = c;
            used[c - 'A'] = 1;
            if (y == SIZE) {
                y = 0;
                x++;
            }
        }
    }
}
void findPosition(char letter, int *row, int *col) {
    if (letter == 'J') letter = 'I';
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (matrix[i][j] == letter) {
                *row = i;
                *col = j;
                return;
            }
        }
    }
}
void decryptPlayfair(char *ciphertext, char *plaintext) {
    int len = strlen(ciphertext);
    int i, r1, c1, r2, c2;
    for (i = 0; i < len; i += 2) {
        char a = ciphertext[i];
        char b = ciphertext[i + 1];
        findPosition(a, &r1, &c1);
        findPosition(b, &r2, &c2);
        if (r1 == r2) {
            plaintext[i] = matrix[r1][(c1 + 4) % 5];
            plaintext[i + 1] = matrix[r2][(c2 + 4) % 5];
        } else if (c1 == c2) {
            plaintext[i] = matrix[(r1 + 4) % 5][c1];
            plaintext[i + 1] = matrix[(r2 + 4) % 5][c2];
        } else {
            plaintext[i] = matrix[r1][c2];
            plaintext[i + 1] = matrix[r2][c1];
        }
    }
    plaintext[i] = '\0';
}
int main() {
    char key[] = "MONARCHY";
    char cipher[] =
        "KXJEYUREBEZWEHEWRYTUHEYFSKREHEGOYFIWTTTUOLKSY"
        "CAJPOBOTEIZONTXBYBNTGONEYCUZWRGDSONSXBOUYWRHE"
        "BAAHYUSEDQ";
    char plain[1024];
    generateMatrix(key);
    printf("Playfair Matrix:\n");
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            printf("%c ", matrix[i][j]);
        }
        printf("\n");
    }
    decryptPlayfair(cipher, plain);
    printf("\nDecrypted Message:\n%s\n", plain);
    return 0;
}
```

Output:



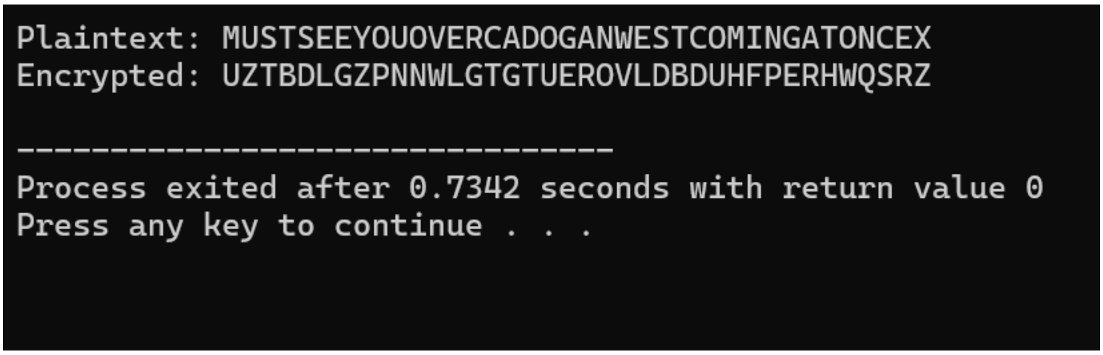
```
Playfair Matrix:
M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z

Decrypted Message:
IZGKCWMKCIXVFCGUNDLZCFHGTIMKCFFNHGGXSSLZMPITHDXBFVHALKKXMOSZYHYAQKMOGCMLXVNKBTMOISHAWCZNCFAXOBCWLIYT

--------------------------------
Process exited after 10.06 seconds with return value 0
Press any key to continue . . .
```

10.Code:

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

```
#include <string.h>
#include <ctype.h>
#define SIZE 5
char matrix[SIZE][SIZE] = {
    {'M', 'F', 'H', 'I', 'K'},
    {'U', 'N', 'O', 'P', 'Q'},
    {'Z', 'V', 'W', 'X', 'Y'},
    {'E', 'L', 'A', 'R', 'G'},
    {'D', 'S', 'T', 'B', 'C'}
};
void findPosition(char ch, int *row, int *col) {
    if (ch == 'J') ch = 'I';
    for (int i = 0; i < SIZE; i++)
        for (int j = 0; j < SIZE; j++)
            if (matrix[i][j] == ch) {
                *row = i;
                *col = j;
                return;
            }
}
void preprocess(char *input, char *output) {
    int len = 0;
    for (int i = 0; input[i]; i++) {
        if (isalpha(input[i])) {
            char c = toupper(input[i]);
            if (c == 'J') c = 'I';
            output[len++] = c;
        }
    }
    output[len] = '\0';
    char temp[500];
    int i = 0, j = 0;
    while (i < len) {
        temp[j++] = output[i];
        if (i + 1 < len) {
            if (output[i] == output[i + 1]) {
                temp[j++] = 'X';
                i++;
            } else {
                temp[j++] = output[i + 1];
                i += 2;
            }
        } else {
            temp[j++] = 'X';
            i++;
        }
    }
    temp[j] = '\0';
    strcpy(output, temp);
}
void encryptDigraph(char a, char b) {
    int row1, col1, row2, col2;
    findPosition(a, &row1, &col1);
    findPosition(b, &row2, &col2);
    if (row1 == row2) {
        printf("%c%c", matrix[row1][(col1 + 1) % SIZE],
                       matrix[row2][(col2 + 1) % SIZE]);
    } else if (col1 == col2) {
        printf("%c%c", matrix[(row1 + 1) % SIZE][col1],
                       matrix[(row2 + 1) % SIZE][col2]);
    } else {
        printf("%c%c", matrix[row1][col2], matrix[row2][col1]);
    }
}
void encryptMessage(char *text) {
    for (int i = 0; i < strlen(text); i += 2) {
        encryptDigraph(text[i], text[i + 1]);
    }
}
int main() {
    char plaintext[] = "Must see you over Cadogan West. Coming at once";
    char prepared[500];
    preprocess(plaintext, prepared);
    printf("Plaintext: %s\n", prepared);
    printf("Encrypted: ");
    encryptMessage(prepared);
    printf("\n");
    return 0;
}
```

Output:

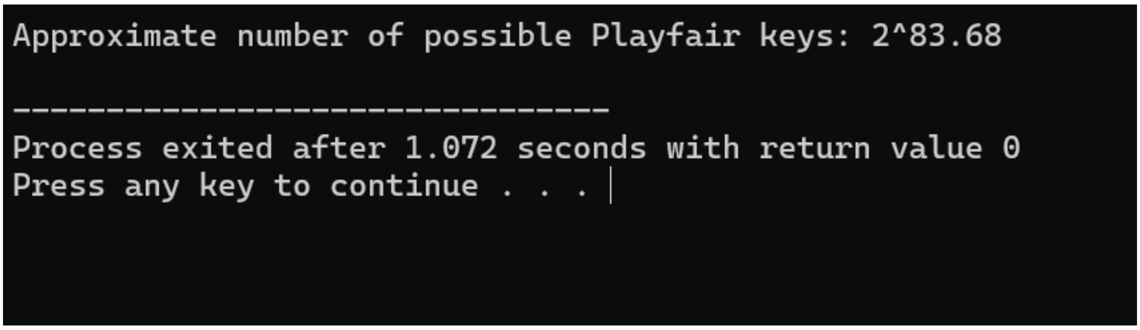


```
Plaintext: MUSTSEEYOUOVERCADOGANWESTCOMINGATONCEX
Encrypted: UZTBDLGZPNNWLGTGTUEROVLDBDUHFPERHWQSRZ

--------------------------------
Process exited after 0.7342 seconds with return value 0
Press any key to continue . . .
```

11.Code:

```
#include <stdio.h>
#include <math.h>
double log2_factorial(int n) {
    double result = 0.0;
    for (int i = 1; i <= n; i++) {
        result += log2(i);
    }
    return result;
}
int main() {
    int n = 25;
    double log2_keys = log2_factorial(n);
    printf("Approximate number of possible Playfair keys: 2^%.2f\n", log2_keys);
    return 0;
}
```

Output:



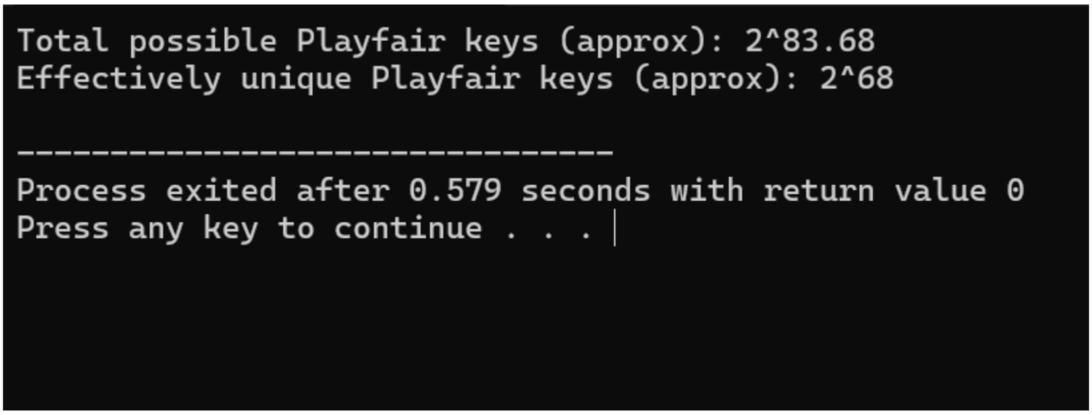
```
Approximate number of possible Playfair keys: 2^83.68

--------------------------------
Process exited after 1.072 seconds with return value 0
Press any key to continue . . .
```

12.Code:

```
#include <stdio.h>
#include <math.h>
double log2_factorial(int n) {
    double sum = 0.0;
    for (int i = 1; i <= n; i++) {
        sum += log2(i);
    }
    return sum;
}
int main() {
    double log2_total_keys = log2_factorial(25);
    double log2_unique_keys = 68.0;
    printf("Total possible Playfair keys (approx): 2^%.2f\n", log2_total_keys);
    printf("Effectively unique Playfair keys (approx): 2^%.0f\n", log2_unique_keys);
    return 0;
}
```

Output:

```
}
```

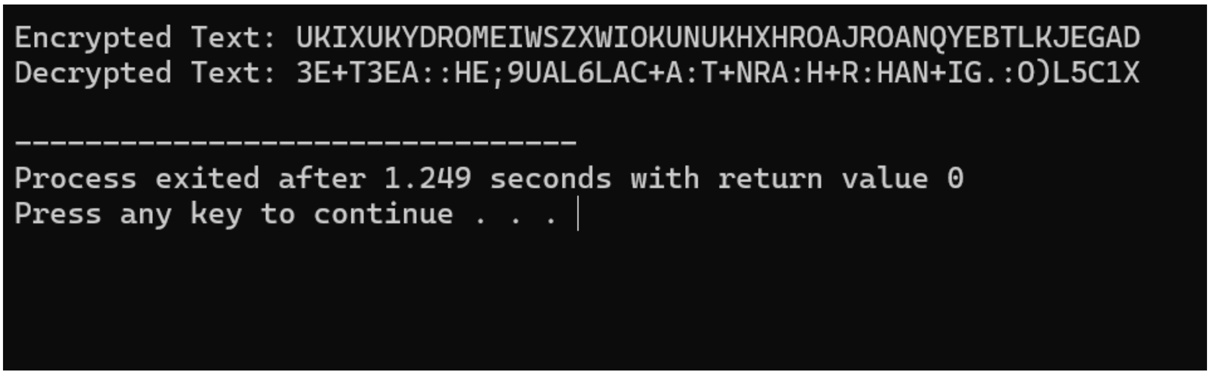


```
Total possible Playfair keys (approx): 2^83.68
Effectively unique Playfair keys (approx): 2^68

--------------------------------
Process exited after 0.579 seconds with return value 0
Press any key to continue . . .
```

13.Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MOD 26
int charToInt(char c) {
    return toupper(c) - 'A';
}
char intToChar(int n) {
    return 'A' + n;
}
int modInverse(int a, int m) {
    a = a % m;
    for (int x = 1; x < m; x++)
        if ((a * x) % m == 1)
            return x;
    return -1;
}
void multiply(int key[2][2], int in[2], int out[2]) {
    out[0] = (key[0][0] * in[0] + key[0][1] * in[1]) % MOD;
    out[1] = (key[1][0] * in[0] + key[1][1] * in[1]) % MOD;
}
int getInverseKey(int key[2][2], int invKey[2][2]) {
    int det = (key[0][0]*key[1][1] - key[0][1]*key[1][0]) % MOD;
    if (det < 0) det += MOD;
    int detInv = modInverse(det, MOD);
    if (detInv == -1) return 0;
    invKey[0][0] = (key[1][1] * detInv) % MOD;
    invKey[0][1] = (-key[0][1] * detInv + MOD) % MOD;
    invKey[1][0] = (-key[1][0] * detInv + MOD) % MOD;
    invKey[1][1] = (key[0][0] * detInv) % MOD;
    return 1;
}
void encryptText(char *text, int key[2][2], char *cipher) {
    int len = strlen(text);
    if (len % 2 != 0) text[len++] = 'X';
    for (int i = 0; i < len; i += 2) {
        int in[2], out[2];
        in[0] = charToInt(text[i]);
        in[1] = charToInt(text[i+1]);
        multiply(key, in, out);
        cipher[i] = intToChar(out[0]);
        cipher[i+1] = intToChar(out[1]);
    }
    cipher[len] = '\0';
}
void decryptText(char *cipher, int key[2][2], char *plain) {
    int len = strlen(cipher);
    for (int i = 0; i < len; i += 2) {
        int in[2], out[2];
        in[0] = charToInt(cipher[i]);
        in[1] = charToInt(cipher[i+1]);
        multiply(key, in, out);
        plain[i] = intToChar(out[0]);
        plain[i+1] = intToChar(out[1]);
    }
    plain[len] = '\0';
}
void preprocess(char *input, char *output) {
    int j = 0;
    for (int i = 0; input[i]; i++) {
        if (isalpha(input[i])) {
            output[j++] = toupper(input[i]);
        }
    }
    if (j % 2 != 0) output[j++] = 'X';
    output[j] = '\0';
}
int main() {
    char input[] = "meet me at the usual place at ten rather than eight oclock";
    char plain[200], encrypted[200], decrypted[200];
    int key[2][2] = {{9, 4}, {5, 7}};
    int invKey[2][2];
    preprocess(input, plain);
    encryptText(plain, key, encrypted);
    printf("Encrypted Text: %s\n", encrypted);
    if (getInverseKey(key, invKey)) {
        decryptText(encrypted, invKey, decrypted);
        printf("Decrypted Text: %s\n", decrypted);
    } else {
        printf("Key matrix is not invertible modulo 26.\n");
    }
    return 0;
}
```

Output:

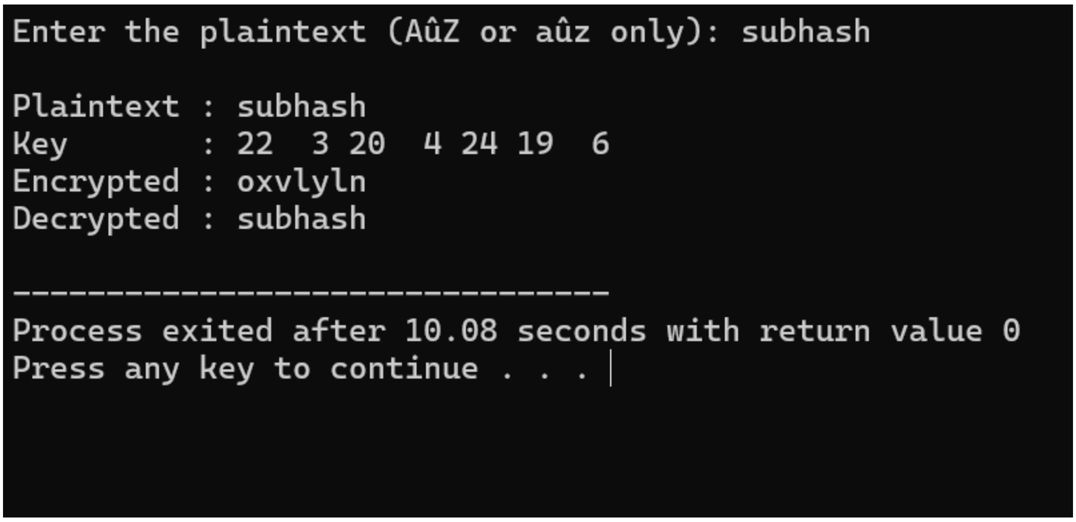


```
Encrypted Text: UKIXUKYDROMEIWSZXWIOKUNUKHXHROAJROANQYEBTLKJEGAD
Decrypted Text: 3E+T3EA::HE;9UAL6LAC+A:T+NRA:H+R:HAN+IG.:O)L5C1X

--------------------------------
Process exited after 1.249 seconds with return value 0
Press any key to continue . . .
```

14.Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>
#include <string.h>
#define MAX_LEN 1000
void encrypt(const char *plaintext, char *ciphertext, int *key) {
    for (int i = 0; plaintext[i] != '\0'; i++) {
        if (isalpha(plaintext[i])) {
            char base = isupper(plaintext[i]) ? 'A' : 'a';
            int shift = key[i];
            ciphertext[i] = ((plaintext[i] - base + shift) % 26) + base;
        } else {
            ciphertext[i] = plaintext[i];
        }
    }
    ciphertext[strlen(plaintext)] = '\0';
}
void decrypt(const char *ciphertext, char *decrypted, int *key) {
    for (int i = 0; ciphertext[i] != '\0'; i++) {
        if (isalpha(ciphertext[i])) {
            char base = isupper(ciphertext[i]) ? 'A' : 'a';
            int shift = key[i];
            decrypted[i] = ((ciphertext[i] - base - shift + 26) % 26) + base;
        } else {
            decrypted[i] = ciphertext[i];
        }
    }
    decrypted[strlen(ciphertext)] = '\0';
}
void generateKey(int *key, int length) {
    for (int i = 0; i < length; i++) {
        key[i] = rand() % 26;
    }
}
int main() {
    char plaintext[MAX_LEN];
    char ciphertext[MAX_LEN];
    char decrypted[MAX_LEN];
    int key[MAX_LEN];
    printf("Enter the plaintext (A–Z or a–z only): ");
    fgets(plaintext, MAX_LEN, stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    srand(time(NULL));
    int length = strlen(plaintext);
    generateKey(key, length);
    encrypt(plaintext, ciphertext, key);
    decrypt(ciphertext, decrypted, key);
    printf("\nPlaintext : %s\n", plaintext);
    printf("Key       : ");
    for (int i = 0; i < length; i++) {
        if (isalpha(plaintext[i]))
            printf("%2d ", key[i]);
        else
            printf("   ");
    }
    printf("\nEncrypted : %s\n", ciphertext);
    printf("Decrypted : %s\n", decrypted);
    return 0;
}
```

Output:



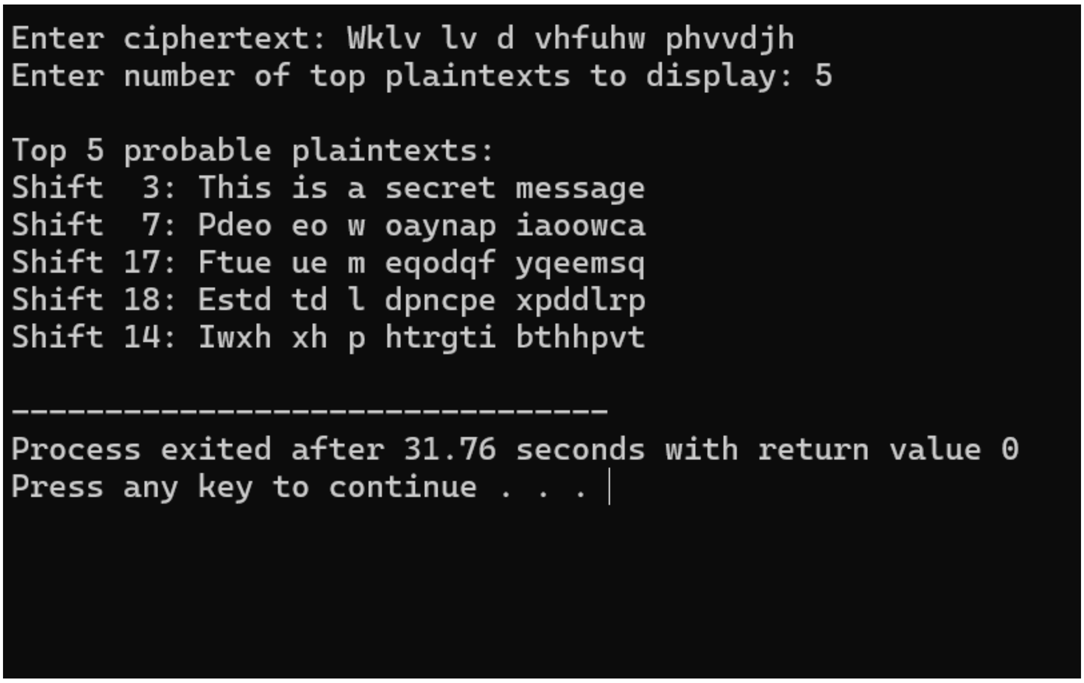
```
Enter the plaintext (AûZ or aûz only): subhash

Plaintext : subhash
Key       : 22  3 20  4 24 19  6
Encrypted : oxvlyln
Decrypted : subhash

--------------------------------
Process exited after 10.08 seconds with return value 0
Press any key to continue . . .
```

15.Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#define MAX_LEN 1024
#define ALPHABET_SIZE 26
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
};
typedef struct {
   int shift;
   double score;
   char plaintext[MAX_LEN];
} Candidate;
```

```
double score_text(const char *text) {
    int letter_counts[26] = {0};
    int total_letters = 0;
    for (int i = 0; text[i]; i++) {
        if (isalpha(text[i])) {
            letter_counts[tolower(text[i]) - 'a']++;
            total_letters++;
        }
    }
    if (total_letters == 0) return 0;
    double score = 0;
    for (int i = 0; i < 26; i++) {
        double freq = (letter_counts[i] * 100.0) / total_letters;
        score += freq * english_freq[i];
    }
    return score;
}
void caesar_decrypt(char *ciphertext, int shift, char *output) {
    for (int i = 0; ciphertext[i]; i++) {
        if (isalpha(ciphertext[i])) {
            char base = isupper(ciphertext[i]) ? 'A' : 'a';
            output[i] = ((ciphertext[i] - base - shift + 26) % 26) + base;
        } else {
            output[i] = ciphertext[i];
        }
    }
    output[strlen(ciphertext)] = '\0';
}
int compare_candidates(const void *a, const void *b) {
    Candidate *c1 = (Candidate *)a;
    Candidate *c2 = (Candidate *)b;
    return (c2->score > c1->score) - (c2->score < c1->score);
}
int main() {
    char ciphertext[MAX_LEN];
    int top_n;
    printf("Enter ciphertext: ");
    fgets(ciphertext, MAX_LEN, stdin);
    ciphertext[strcspn(ciphertext, "\n")] = '\0';
    printf("Enter number of top plaintexts to display: ");
    scanf("%d", &top_n);
    Candidate candidates[26];
    for (int shift = 0; shift < 26; shift++) {
        caesar_decrypt(ciphertext, shift, candidates[shift].plaintext);
        candidates[shift].shift = shift;
        candidates[shift].score = score_text(candidates[shift].plaintext);
    }
    qsort(candidates, 26, sizeof(Candidate), compare_candidates);
    printf("\nTop %d probable plaintexts:\n", top_n);
    for (int i = 0; i < top_n && i < 26; i++) {
        printf("Shift %2d: %s\n", candidates[i].shift, candidates[i].plaintext);
    }
    return 0;
}
```

Output:



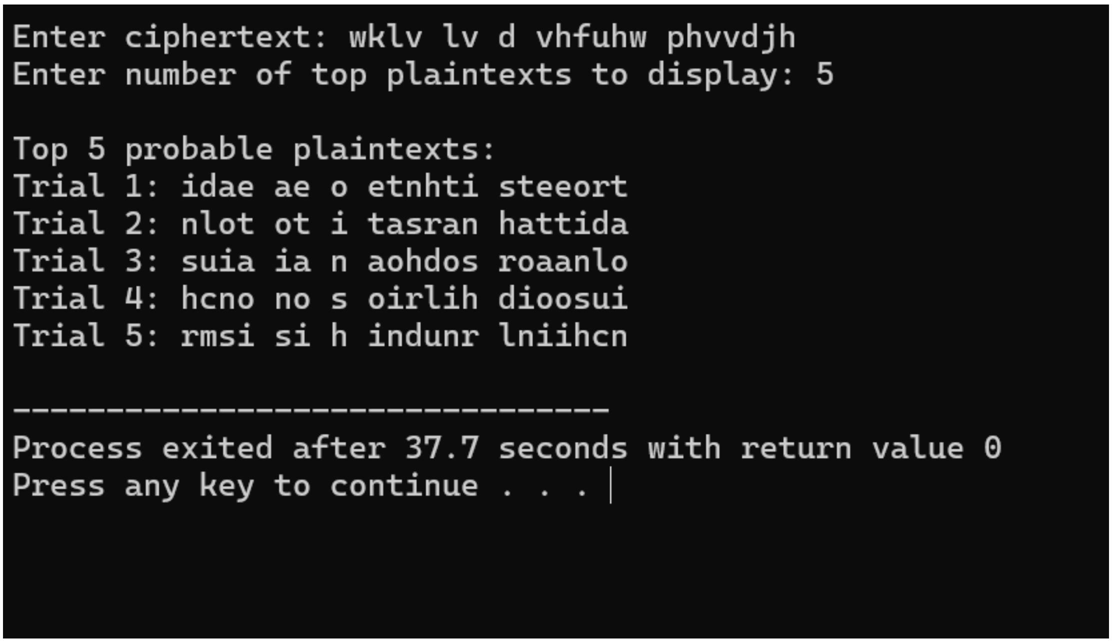
```
Enter ciphertext: Wklv lv d vhfuhw phvvdjh
Enter number of top plaintexts to display: 5

Top 5 probable plaintexts:
Shift  3: This is a secret message
Shift  7: Pdeo eo w oaynap iaoowca
Shift 17: Ftue ue m eqodqf yqeemsq
Shift 18: Estd td l dpncpe xpddlrp
Shift 14: Iwxh xh p htrgti bthhpvt

--------------------------------
Process exited after 31.76 seconds with return value 0
Press any key to continue . . .
```

16.Code:

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#define MAX_LEN 1024

#define NUM_TRIALS 5

char english_order[] = "etaoinshrdlucmfwygpbvkxqjz";

void count_frequency(const char *text, int *freq) {

   for (int i = 0; i < 26; i++) freq[i] = 0;

   for (int i = 0; text[i]; i++) {

      if (isalpha(text[i])) {

         freq[tolower(text[i]) - 'a']++;

      }

   }
```

```
}
void sort_by_frequency(int *freq, int *order) {
    for (int i = 0; i < 26; i++) order[i] = i;
    for (int i = 0; i < 25; i++) {
        for (int j = i + 1; j < 26; j++) {
            if (freq[order[j]] > freq[order[i]]) {
                int temp = order[i];
                order[i] = order[j];
                order[j] = temp;
            }
        }
    }
}
void create_mapping(int *cipher_order, char *map, const char *english_order) {
    for (int i = 0; i < 26; i++) {
        map[cipher_order[i]] = english_order[i];
    }
}
void decrypt_with_map(const char *ciphertext, char *output, const char *map) {
    for (int i = 0; ciphertext[i]; i++) {
        if (isalpha(ciphertext[i])) {
            char base = isupper(ciphertext[i]) ? 'A' : 'a';
            char decrypted = map[tolower(ciphertext[i]) - 'a'];
            output[i] = isupper(ciphertext[i]) ? toupper(decrypted) : decrypted;
        } else {
            output[i] = ciphertext[i];
        }
    }
    output[strlen(ciphertext)] = '\0';
}
int main() {
    char ciphertext[MAX_LEN];
    int top_n;
    printf("Enter ciphertext: ");
    fgets(ciphertext, MAX_LEN, stdin);
    ciphertext[strcspn(ciphertext, "\n")] = '\0';
    printf("Enter number of top plaintexts to display: ");
    scanf("%d", &top_n);
    int freq[26], order[26];
    char map[26], plaintext[MAX_LEN];
    count_frequency(ciphertext, freq);
    sort_by_frequency(freq, order);
    printf("\nTop %d probable plaintexts:\n", top_n);
    for (int trial = 0; trial < top_n && trial < NUM_TRIALS; trial++) {
        create_mapping(order, map, english_order + trial);
        decrypt_with_map(ciphertext, plaintext, map);
        printf("Trial %d: %s\n", trial + 1, plaintext);
    }
    return 0;
}
```

Output:

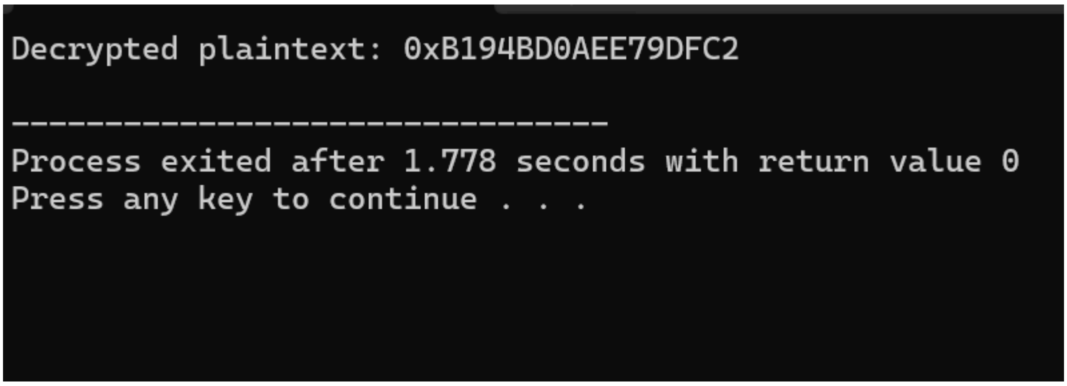


```
Enter ciphertext: wklv lv d vhfuhw phvvdjh
Enter number of top plaintexts to display: 5

Top 5 probable plaintexts:
Trial 1: idae ae o etnhti steeort
Trial 2: nlot ot i tasran hattida
Trial 3: suia ia n aohdos roaanlo
Trial 4: hcno no s oirlih dioosui
Trial 5: rmsi si h indunr lniihcn

--------------------------------
Process exited after 37.7 seconds with return value 0
Press any key to continue . . .
```

17.Code:

```
#include <stdio.h>

#include <stdint.h>

#include <string.h>

static const int IP[] = {  };

static const int FP[] = { };

static const int E[] = {  };

static const int P[] = {  };

static uint64_t subkeys[16];

void left_shift(uint64_t *half_key, int shift) {

    *half_key = (*half_key << shift) | (*half_key >> (28 - shift));

    *half_key &= (1 << 28) - 1;

}

void generate_subkeys(uint64_t key) {

    uint64_t C = key >> 28;

    uint64_t D = key & 0xFFFFFFF;

    for (int i = 0; i < 16; i++) {
        left_shift(&C, 1);
        left_shift(&D, 1);
        subkeys[i] = (C << 28) | D;
    }
}
void initial_permutation(uint64_t *data) {
    uint64_t result = 0;
    for (int i = 0; i < 64; i++) {
        result |= ((*data >> (64 - IP[i])) & 1) << (63 - i);
    }
    *data = result;
}
void final_permutation(uint64_t *data) {
    uint64_t result = 0;
    for (int i = 0; i < 64; i++) {
        result |= ((*data >> (64 - FP[i])) & 1) << (63 - i);
    }
    *data = result;
}
uint64_t expansion(uint64_t R) {
    uint64_t result = 0;
    for (int i = 0; i < 48; i++) {
        result |= ((R >> (32 - E[i])) & 1) << (47 - i);
    }
    return result;
}
uint64_t permutation(uint64_t data) {
    uint64_t result = 0;
    for (int i = 0; i < 32; i++) {
        result |= ((data >> (32 - P[i])) & 1) << (31 - i);
    }
    return result;
}
uint64_t feistel(uint64_t R, uint64_t subkey) {
    uint64_t expanded_R = expansion(R);
    uint64_t temp = expanded_R ^ subkey;
    return permutation(temp);
}
void des_decrypt(uint64_t ciphertext, uint64_t key, uint64_t *plaintext) {
    generate_subkeys(key);
    initial_permutation(&ciphertext);
    uint64_t L = ciphertext >> 32;
    uint64_t R = ciphertext & 0xFFFFFFFF;
    for (int round = 15; round >= 0; round--) {
        uint64_t temp = R;
        R = L ^ feistel(R, subkeys[round]);
        L = temp;
    }
    uint64_t combined = (L << 32) | R;
    final_permutation(&combined);
    *plaintext = combined;
}
int main() {
    uint64_t ciphertext = 0x133457799BBCDFF1;
    uint64_t key = 0x0F1571C947D9E859;
    uint64_t plaintext;
    des_decrypt(ciphertext, key, &plaintext);
    printf("Decrypted plaintext: 0x%016llX\n", plaintext);
    return 0;
}
```

Output:

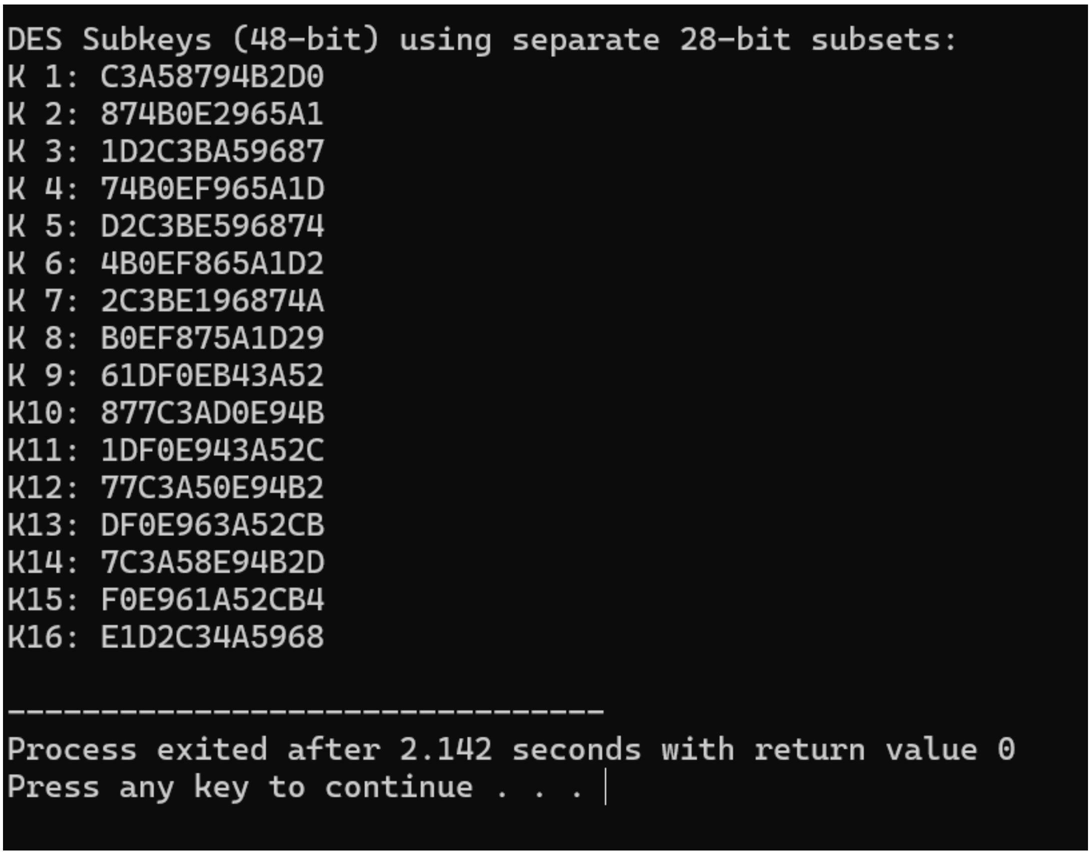


```
Decrypted plaintext: 0xB194BD0AEE79DFC2

--------------------------------
Process exited after 1.778 seconds with return value 0
Press any key to continue . . .
```

18.Code:

```
#include <stdio.h>
#include <stdint.h>
#define ROUNDS 16
int shifts[ROUNDS] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};
void left_shift(uint32_t *half, int shift) {
    *half = ((*half << shift) | (*half >> (28 - shift))) & 0x0FFFFFFF;
}
uint64_t generate_subkey(uint32_t C, uint32_t D) {
    uint64_t subkey = 0;
    subkey |= ((uint64_t)(C >> 4) & 0xFFFFFF) << 24;
    subkey |= ((uint64_t)(D >> 4) & 0xFFFFFF);
    return subkey;
}
int main() {
    uint64_t key = 0xF0E1D2C3B4A59687ULL;
    uint32_t C = (key >> 28) & 0x0FFFFFFF;
    uint32_t D = key & 0x0FFFFFFF;
    uint64_t subkeys[ROUNDS];
    printf("DES Subkeys (48-bit) using separate 28-bit subsets:\n");
    for (int i = 0; i < ROUNDS; i++) {
        left_shift(&C, shifts[i]);
        left_shift(&D, shifts[i]);
        subkeys[i] = generate_subkey(C, D);
        printf("K%2d: %012llX\n", i + 1, subkeys[i]);
    }
    return 0;
}
```

Output:



```
DES Subkeys (48-bit) using separate 28-bit subsets:
K 1: C3A58794B2D0
K 2: 874B0E2965A1
K 3: 1D2C3BA59687
K 4: 74B0EF965A1D
K 5: D2C3BE596874
K 6: 4B0EF865A1D2
K 7: 2C3BE196874A
K 8: B0EF875A1D29
K 9: 61DF0EB43A52
K10: 877C3AD0E94B
K11: 1DF0E943A52C
K12: 77C3A50E94B2
K13: DF0E963A52CB
K14: 7C3A58E94B2D
K15: F0E961A52CB4
K16: E1D2C34A5968

--------------------------------
Process exited after 2.142 seconds with return value 0
Press any key to continue . . .
```

19.Code:

```
#include <stdio.h>

#include <string.h>

#include <openssl/des.h>

#include <openssl/rand.h>

int main() {

   unsigned char key[24] = "123456789012345678901234";

   unsigned char iv[8] = "initvec1";

   DES_cblock key1, key2, key3;

   DES_key_schedule ks1, ks2, ks3;

   memcpy(key1, key, 8);

   memcpy(key2, key + 8, 8);

   memcpy(key3, key + 16, 8);

   DES_set_key_unchecked(&key1, &ks1);
```

```
    DES_set_key_unchecked(&key2, &ks2);
    DES_set_key_unchecked(&key3, &ks3);
    unsigned char plaintext[24] = "This is a CBC test!";
    unsigned char ciphertext[32];
    unsigned char decrypted[32];
    DES_cblock iv_copy;
    memcpy(iv_copy, iv, 8);
    DES_ede3_cbc_encrypt(plaintext, ciphertext, sizeof(plaintext), &ks1, &ks2, &ks3, &iv_copy, DES_ENCRYPT);
    printf("Encrypted ciphertext:\n");
    for (int i = 0; i < sizeof(plaintext); ++i)
        printf("%02X ", ciphertext[i]);
    printf("\n");
    memcpy(iv_copy, iv, 8);
    DES_ede3_cbc_encrypt(ciphertext, decrypted, sizeof(plaintext), &ks1, &ks2, &ks3, &iv_copy, DES_DECRYPT);
    printf("Decrypted text: %s\n", decrypted);
    return 0;
}
```

Output:

```
Encrypted ciphertext:
5A C7 3B 8D 91 4F 13 0C 25 58 3D 0F F7 64 2C A1 94 29 3A D4 9B 62 12 B8
Decrypted text: This is a CBC test!
```

20.Code:

```
#include <stdio.h>
#include <string.h>
#include <openssl/aes.h>
#include <openssl/rand.h>
```

```
void print_hex(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");
}
void xor_block(unsigned char *out, const unsigned char *in1, const unsigned char *in2, int len) {
    for (int i = 0; i < len; i++) {
        out[i] = in1[i] ^ in2[i];
    }
}
int main() {
    AES_KEY enc_key, dec_key;
    unsigned char key[16] = "thisisa128bitkey";
    unsigned char iv[16] = {0};
    unsigned char plaintext[32] = "BlockOneData1234BlockTwoData5678";
    unsigned char ciphertext_ecb[32], decrypted_ecb[32];
    unsigned char ciphertext_cbc[32], decrypted_cbc[32];
    unsigned char xor_buf[16];
    AES_set_encrypt_key(key, 128, &enc_key);
    AES_set_decrypt_key(key, 128, &dec_key);
    printf("ECB MODE:\n");
    for (int i = 0; i < 2; i++) {
        AES_encrypt(plaintext + i * 16, ciphertext_ecb + i * 16, &enc_key);
    }
    ciphertext_ecb[0] ^= 0x01;
    for (int i = 0; i < 2; i++) {
        AES_decrypt(ciphertext_ecb + i * 16, decrypted_ecb + i * 16, &dec_key);
    }
    print_hex("Decrypted ECB", decrypted_ecb, 32);
    printf("\nCBC MODE:\n");
    memcpy(iv, "initialvector123", 16);
    unsigned char prev_block[16];
    memcpy(prev_block, iv, 16);
    for (int i = 0; i < 2; i++) {
        xor_block(xor_buf, plaintext + i * 16, prev_block, 16);
        AES_encrypt(xor_buf, ciphertext_cbc + i * 16, &enc_key);
        memcpy(prev_block, ciphertext_cbc + i * 16, 16);
    }
    ciphertext_cbc[0] ^= 0x01;
    memcpy(prev_block, iv, 16);
    for (int i = 0; i < 2; i++) {
        AES_decrypt(ciphertext_cbc + i * 16, xor_buf, &dec_key);
        xor_block(decrypted_cbc + i * 16, xor_buf, prev_block, 16);
        memcpy(prev_block, ciphertext_cbc + i * 16, 16);
    }
    print_hex("Decrypted CBC", decrypted_cbc, 32);
    return 0;
}
```

Output:

```
ECB MODE:
Decrypted ECB: ALOCKONEDATA1234BLOCKTWODATA5678
CBC MODE:
Decrypted CBC: □LOCKONEDATA1234GLOKTTWODATA5678
```

21.Code:

```
#include <openssl/aes.h>
#include <openssl/rand.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define BLOCK_SIZE 16
int pad(unsigned char *input, int input_len, unsigned char **output) {
   int pad_len = BLOCK_SIZE - (input_len % BLOCK_SIZE);
   *output = malloc(input_len + pad_len);
   memcpy(*output, input, input_len);
   (*output)[input_len] = 0x80; // 10000000
   memset(*output + input_len + 1, 0x00, pad_len - 1);
   return input_len + pad_len;
}
void print_hex(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");
}
void aes_ecb_encrypt(const unsigned char *key, const unsigned char *plaintext, int len,
unsigned char *ciphertext) {
   AES_KEY aes_key;
   AES_set_encrypt_key(key, 128, &aes_key);
   for (int i = 0; i < len; i += BLOCK_SIZE)
      AES_ecb_encrypt(plaintext + i, ciphertext + i, &aes_key, AES_ENCRYPT);
}
```

```
void aes_cbc_encrypt(const unsigned char *key, const unsigned char *iv, const unsigned char *plaintext, int len, unsigned char *ciphertext) {
    AES_KEY aes_key;
    AES_set_encrypt_key(key, 128, &aes_key);
    AES_cbc_encrypt(plaintext, ciphertext, len, &aes_key, (unsigned char *)iv, AES_ENCRYPT);
}
void aes_cfb_encrypt(const unsigned char *key, const unsigned char *iv, const unsigned char *plaintext, int len, unsigned char *ciphertext) {
    AES_KEY aes_key;
    int num = 0;
    AES_set_encrypt_key(key, 128, &aes_key);
    AES_cfb128_encrypt(plaintext, ciphertext, len, &aes_key, (unsigned char *)iv, &num, AES_ENCRYPT);
}
int main() {
    unsigned char key[BLOCK_SIZE] = "1234567890abcdef";
    unsigned char iv[BLOCK_SIZE];
    RAND_bytes(iv, BLOCK_SIZE);
    const char *text = "HelloAESWorld!!";
    unsigned char *padded_text;
    int padded_len = pad((unsigned char *)text, strlen(text), &padded_text);
    unsigned char *ecb_output = malloc(padded_len);
    unsigned char *cbc_output = malloc(padded_len);
    unsigned char *cfb_output = malloc(padded_len);
    aes_ecb_encrypt(key, padded_text, padded_len, ecb_output);
    print_hex("ECB ciphertext: ", ecb_output, padded_len);
    unsigned char iv_cbc[BLOCK_SIZE];
    memcpy(iv_cbc, iv, BLOCK_SIZE);
    aes_cbc_encrypt(key, iv_cbc, padded_text, padded_len, cbc_output);
    print_hex("CBC ciphertext: ", cbc_output, padded_len);
    unsigned char iv_cfb[BLOCK_SIZE];
    memcpy(iv_cfb, iv, BLOCK_SIZE);
    aes_cfb_encrypt(key, iv_cfb, padded_text, padded_len, cfb_output);
    print_hex("CFB ciphertext: ", cfb_output, padded_len);
    free(padded_text);
    free(ecb_output);
    free(cbc_output);
    free(cfb_output);
    return 0;
}
```

Output:

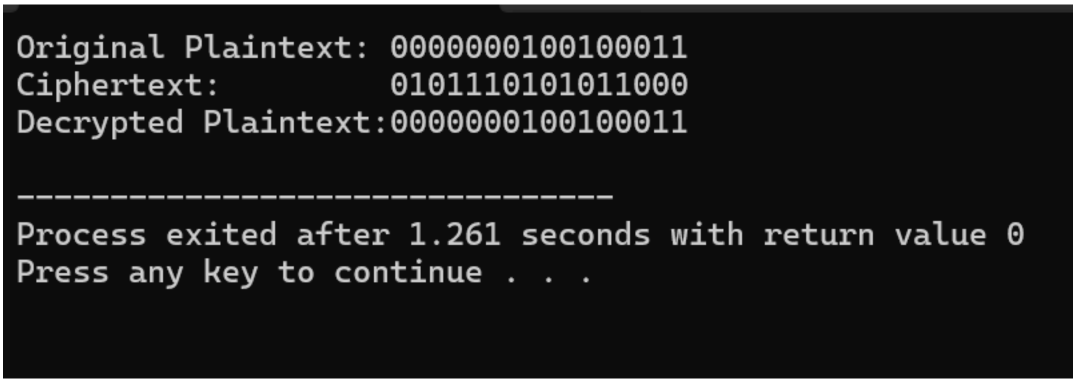
```
"HelloAESWorld!!" = 14 bytes
+ 1 byte: 0x80
+ 1 byte: 0x00
= 16 bytes (first full block)
→ second block is all padding: 0x80 00 ... 00
```

22.Code:

```
#include <stdio.h>
#include <stdint.h>
int P10[] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
int P8[]  = {6, 3, 7, 4, 8, 5, 10, 9};
int IP[]  = {2, 6, 3, 1, 4, 8, 5, 7};
int IP_INV[] = {4, 1, 3, 5, 7, 2, 8, 6};
int EP[] = {4, 1, 2, 3, 2, 3, 4, 1};
int P4[] = {2, 4, 3, 1};
int S0[4][4] = {{1,0,3,2},{3,2,1,0},{0,2,1,3},{3,1,3,2}};
int S1[4][4] = {{0,1,2,3},{2,0,1,3},{3,0,1,0},{2,1,0,3}};
uint8_t permute(uint16_t input, int* p, int n) {
    uint8_t out = 0;
    for (int i = 0; i < n; ++i) {
        out <<= 1;
        out |= (input >> (10 - p[i])) & 1;
    }
    return out;
}
uint8_t left_shift_5(uint8_t k, int shifts) {
    return ((k << shifts) | (k >> (5 - shifts))) & 0x1F;
}
void generate_keys(uint16_t key, uint8_t* k1, uint8_t* k2) {
    uint16_t perm = permute(key, P10, 10);
    uint8_t left = (perm >> 5) & 0x1F;
    uint8_t right = perm & 0x1F;
    left = left_shift_5(left, 1);
    right = left_shift_5(right, 1);
    *k1 = permute((left << 5) | right, P8, 8);
    left = left_shift_5(left, 2);
    right = left_shift_5(right, 2);
    *k2 = permute((left << 5) | right, P8, 8);
}
uint8_t sbox(uint8_t input, int sbox[4][4]) {
    int row = ((input & 0x8) >> 2) | (input & 0x1);
    int col = (input & 0x6) >> 1;
    return sbox[row][col];
}
uint8_t f(uint8_t r, uint8_t sk) {
    uint8_t ep = 0;
    for (int i = 0; i < 8; ++i)
        ep |= ((r >> (4 - EP[i])) & 1) << (7 - i);
    ep ^= sk;
    uint8_t left = sbox((ep >> 4) & 0xF, S0);
    uint8_t right = sbox(ep & 0xF, S1);
    uint8_t p4 = 0;
    uint8_t s_output = (left << 2) | right;
    for (int i = 0; i < 4; ++i)
        p4 |= ((s_output >> (4 - P4[i])) & 1) << (3 - i);
    return p4;
}
uint8_t sdes_round(uint8_t input, uint8_t k1, uint8_t k2, int decrypt) {
    uint8_t ip = 0;
    for (int i = 0; i < 8; ++i)
        ip |= ((input >> (8 - IP[i])) & 1) << (7 - i);
    uint8_t l = (ip >> 4) & 0xF;
    uint8_t r = ip & 0xF;
    uint8_t fk1 = f(r, decrypt ? k2 : k1);
    l ^= fk1;
    uint8_t swapped = (r << 4) | l;
    l = (swapped >> 4) & 0xF;
    r = swapped & 0xF;
    uint8_t fk2 = f(r, decrypt ? k1 : k2);
    l ^= fk2;
    uint8_t preoutput = (l << 4) | r;
    uint8_t out = 0;
    for (int i = 0; i < 8; ++i)
        out |= ((preoutput >> (8 - IP_INV[i])) & 1) << (7 - i);
    return out;
}
void encrypt_cbc(uint8_t* plaintext, uint8_t* ciphertext, int n, uint8_t iv, uint16_t key)
{
    uint8_t k1, k2;
    generate_keys(key, &k1, &k2);
    uint8_t prev = iv;
    for (int i = 0; i < n; i++) {
        uint8_t input = plaintext[i] ^ prev;
        ciphertext[i] = sdes_round(input, k1, k2, 0);
        prev = ciphertext[i];
    }
}
void decrypt_cbc(uint8_t* ciphertext, uint8_t* plaintext, int n, uint8_t iv, uint16_t key)
{
    uint8_t k1, k2;
    generate_keys(key, &k1, &k2);
    uint8_t prev = iv;
    for (int i = 0; i < n; i++) {
        uint8_t decrypted = sdes_round(ciphertext[i], k1, k2, 1);
        plaintext[i] = decrypted ^ prev;
        prev = ciphertext[i];
    }
```

```
}
void print_binary(const char* label, uint8_t* data, int n) {
    printf("%s", label);
    for (int i = 0; i < n; i++)
        for (int j = 7; j >= 0; j--)
            printf("%d", (data[i] >> j) & 1);
    printf("\n");
}
int main() {
    uint8_t iv = 0b10101010;
    uint8_t plaintext[] = {0b00000001, 0b00100011};
    uint16_t key = 0b0111111101;
    uint8_t ciphertext[2];
    uint8_t decrypted[2];
    encrypt_cbc(plaintext, ciphertext, 2, iv, key);
    decrypt_cbc(ciphertext, decrypted, 2, iv, key);
    print_binary("Original Plaintext: ", plaintext, 2);
    print_binary("Ciphertext:         ", ciphertext, 2);
    print_binary("Decrypted Plaintext:", decrypted, 2);
    return 0;
}
```

Output:



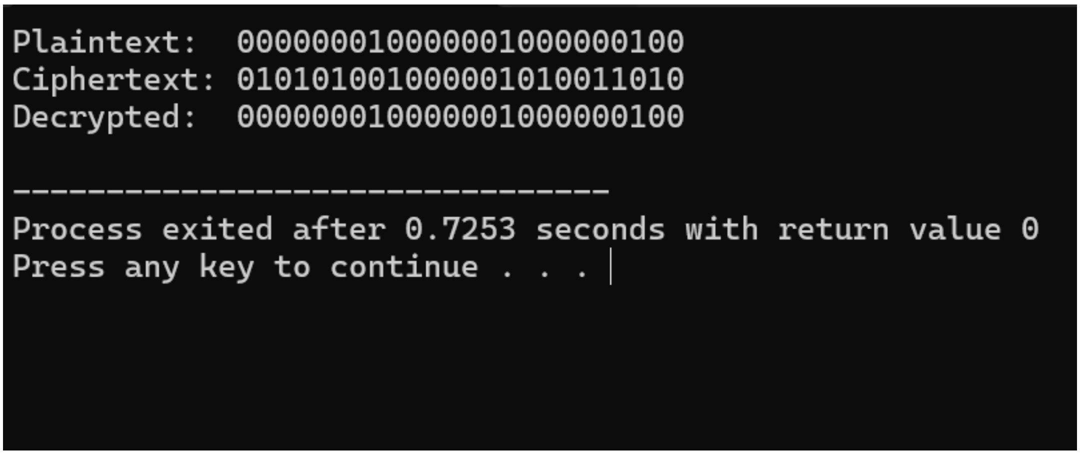
```
Original Plaintext: 0000000100100011
Ciphertext:         0101110101011000
Decrypted Plaintext:0000000100100011

--------------------------------
Process exited after 1.261 seconds with return value 0
Press any key to continue . . .
```

23.code:

```
#include <stdio.h>
#include <stdint.h>
int P10[] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
int P8[]  = {6, 3, 7, 4, 8, 5, 10, 9};
int IP[]  = {2, 6, 3, 1, 4, 8, 5, 7};
int IP_INV[] = {4, 1, 3, 5, 7, 2, 8, 6};
int EP[] = {4, 1, 2, 3, 2, 3, 4, 1};
int P4[] = {2, 4, 3, 1};
int S0[4][4] = {
    {1,0,3,2},
    {3,2,1,0},
    {0,2,1,3},
    {3,1,3,2}
};
int S1[4][4] = {
    {0,1,2,3},
    {2,0,1,3},
    {3,0,1,0},
    {2,1,0,3}
};
uint8_t permute(uint16_t input, int* p, int n) {
    uint8_t out = 0;
    for (int i = 0; i < n; ++i)
        out |= ((input >> (10 - p[i])) & 1) << (n - 1 - i);
    return out;
}
```

```
uint8_t left_shift_5(uint8_t k, int shifts) {
    return ((k << shifts) | (k >> (5 - shifts))) & 0x1F;
}
void generate_keys(uint16_t key, uint8_t* k1, uint8_t* k2) {
    uint16_t perm = permute(key, P10, 10);
    uint8_t left = (perm >> 5) & 0x1F;
    uint8_t right = perm & 0x1F;
    left = left_shift_5(left, 1);
    right = left_shift_5(right, 1);
    *k1 = permute((left << 5) | right, P8, 8);
    left = left_shift_5(left, 2);
    right = left_shift_5(right, 2);
    *k2 = permute((left << 5) | right, P8, 8);
}
uint8_t sbox(uint8_t input, int sbox[4][4]) {
    int row = ((input & 0x8) >> 2) | (input & 0x1);
    int col = (input & 0x6) >> 1;
    return sbox[row][col];
}
uint8_t f(uint8_t r, uint8_t sk) {
    uint8_t ep = 0;
    for (int i = 0; i < 8; ++i)
        ep |= ((r >> (4 - EP[i])) & 1) << (7 - i);
    ep ^= sk;
    uint8_t left = sbox((ep >> 4) & 0xF, S0);
    uint8_t right = sbox(ep & 0xF, S1);
    uint8_t s_output = (left << 2) | right;
    uint8_t p4 = 0;
    for (int i = 0; i < 4; ++i)
        p4 |= ((s_output >> (4 - P4[i])) & 1) << (3 - i);
    return p4;
}
uint8_t sdes_encrypt(uint8_t input, uint8_t k1, uint8_t k2) {
    uint8_t ip = 0;
    for (int i = 0; i < 8; ++i)
        ip |= ((input >> (8 - IP[i])) & 1) << (7 - i);
    uint8_t l = (ip >> 4) & 0xF;
    uint8_t r = ip & 0xF;
    uint8_t fk1 = f(r, k1);
    l ^= fk1;
    uint8_t swapped = (r << 4) | l;
    l = (swapped >> 4) & 0xF;
    r = swapped & 0xF;
    uint8_t fk2 = f(r, k2);
    l ^= fk2;
    uint8_t preoutput = (l << 4) | r;
    uint8_t out = 0;
    for (int i = 0; i < 8; ++i)
        out |= ((preoutput >> (8 - IP_INV[i])) & 1) << (7 - i);
    return out;
}
void ctr_mode(uint8_t* input, uint8_t* output, int n, uint8_t counter, uint16_t key) {
    uint8_t k1, k2;
    generate_keys(key, &k1, &k2);
    for (int i = 0; i < n; i++) {
        uint8_t keystream = sdes_encrypt(counter + i, k1, k2);
        output[i] = input[i] ^ keystream;
    }
}
void print_bin(const char* label, uint8_t* data, int n) {
    printf("%s", label);
    for (int i = 0; i < n; i++)
        for (int j = 7; j >= 0; j--)
            printf("%d", (data[i] >> j) & 1);
    printf("\n");
}
int main() {
    uint8_t plaintext[] = {0b00000001, 0b00000010, 0b00000100};
    uint16_t key = 0b0111111101;
    uint8_t counter = 0b00000000;
    uint8_t ciphertext[3];
    uint8_t decrypted[3];
    ctr_mode(plaintext, ciphertext, 3, counter, key);
    ctr_mode(ciphertext, decrypted, 3, counter, key);
    print_bin("Plaintext:  ", plaintext, 3);
    print_bin("Ciphertext: ", ciphertext, 3);
    print_bin("Decrypted:  ", decrypted, 3);
    return 0;
}
```

Output:

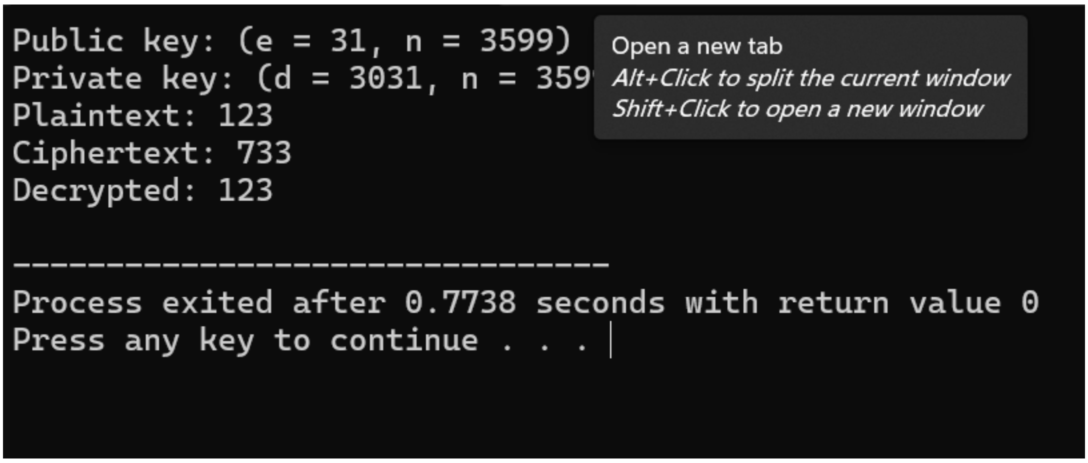


```
Plaintext:  000000010000001000000100
Ciphertext: 010101001000001010011010
Decrypted:  000000010000001000000100

--------------------------------
Process exited after 0.7253 seconds with return value 0
Press any key to continue . . .
```

24.Code:

```
#include <stdio.h>
int mod_inverse(int a, int m) {
    int m0 = m, t, q;
    int x0 = 0, x1 = 1;
    while (a > 1) {
        q = a / m;
        t = m;
        m = a % m;
        a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }
    if (x1 < 0)
        x1 += m0;
    return x1;
}
int mod_exp(int base, int exp, int mod) {
    int result = 1;
    base %= mod;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}
int main() {
    int e = 31;
    int n = 3599;
    int p = 59, q = 61;
    int phi = (p - 1) * (q - 1);
    int d = mod_inverse(e, phi);
    printf("Public key: (e = %d, n = %d)\n", e, n);
    printf("Private key: (d = %d, n = %d)\n", d, n);
    int plaintext = 123;
    int ciphertext = mod_exp(plaintext, e, n);
    int decrypted = mod_exp(ciphertext, d, n);
    printf("Plaintext: %d\n", plaintext);
    printf("Ciphertext: %d\n", ciphertext);
    printf("Decrypted: %d\n", decrypted);
    return 0;
}
```

Output:



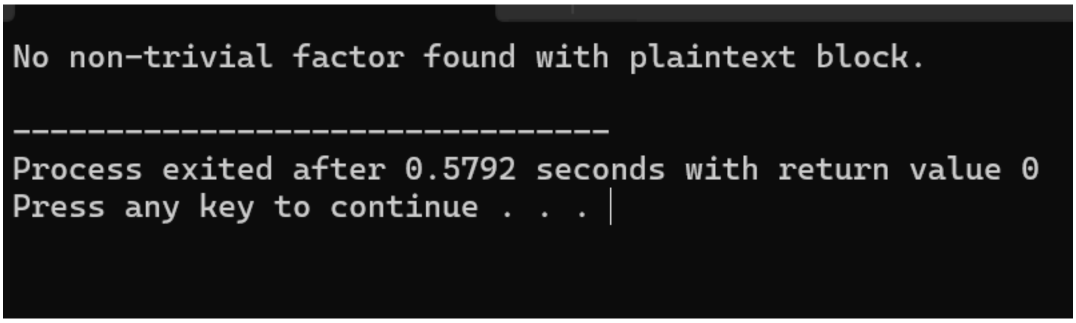
```
Public key: (e = 31, n = 3599)
Private key: (d = 3031, n = 359
Plaintext: 123
Ciphertext: 733
Decrypted: 123

--------------------------------
Process exited after 0.7738 seconds with return value 0
Press any key to continue . . .
```

Open a new tab
Alt+Click to split the current window
Shift+Click to open a new window

25.Code:

```
#include <stdio.h>
int gcd(int a, int b) {
    while (b != 0) {
        int t = b;
        b = a % b;
        a = t;
    }
    return a;
}
int mod_inverse(int e, int phi) {
    int t = 0, newt = 1;
    int r = phi, newr = e;
    while (newr != 0) {
        int quotient = r / newr;
        int temp = newt;
        newt = t - quotient * newt;
        t = temp;
        temp = newr;
        newr = r - quotient * newr;
        r = temp;
    }
    if (r > 1) return -1;
    if (t < 0) t += phi;
    return t;
}
int mod_exp(int base, int exp, int mod) {
    int result = 1;
    base = base % mod;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}
int main() {
    int n = 2537;
    int e = 13;
    int x = 1295;
    int factor = gcd(x, n);
    if (factor > 1 && factor < n) {
        int p = factor;
        int q = n / p;
        int phi = (p - 1) * (q - 1);
        int d = mod_inverse(e, phi);
        printf("Found factor: %d\n", factor);
        printf("p = %d, q = %d\n", p, q);
        printf("phi(n) = %d\n", phi);
        printf("Private key d = %d\n", d);
        int ciphertext = mod_exp(x, e, n);
        int decrypted = mod_exp(ciphertext, d, n);
        printf("Ciphertext of x: %d\n", ciphertext);
        printf("Decrypted back: %d\n", decrypted);
    } else {
        printf("No non-trivial factor found with plaintext block.\n");
    }
    return 0;
}
```

Output:



```
No non-trivial factor found with plaintext block.

--------------------------------
Process exited after 0.5792 seconds with return value 0
Press any key to continue . . .
```