1. **CAESAR CIPHER**

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

void caesar_cipher(char *text, int k) {
for (int i = 0; text[i]; i++)
if (isalpha(text[i]))
text[i] = ((text[i] - (isupper(text[i]) ? 'A' : 'a') + k) % 26) + (isupper(text[i]) ? 'A' : 'a');
}

int main() {
char text[100]; int k;
printf("Enter text: "); fgets(text, sizeof(text), stdin); text[strcspn(text, "\n")] = 0;
printf("Enter shift (1-25): "); scanf("%d", &k);
caesar_cipher(text, k);
printf("Encrypted text: %s\n", text);
return 0;
}
```

```
Output                                                    Clear
Enter text: gowthami
Enter shift (1-25): 3
Encrypted text: jrzwkdpl


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

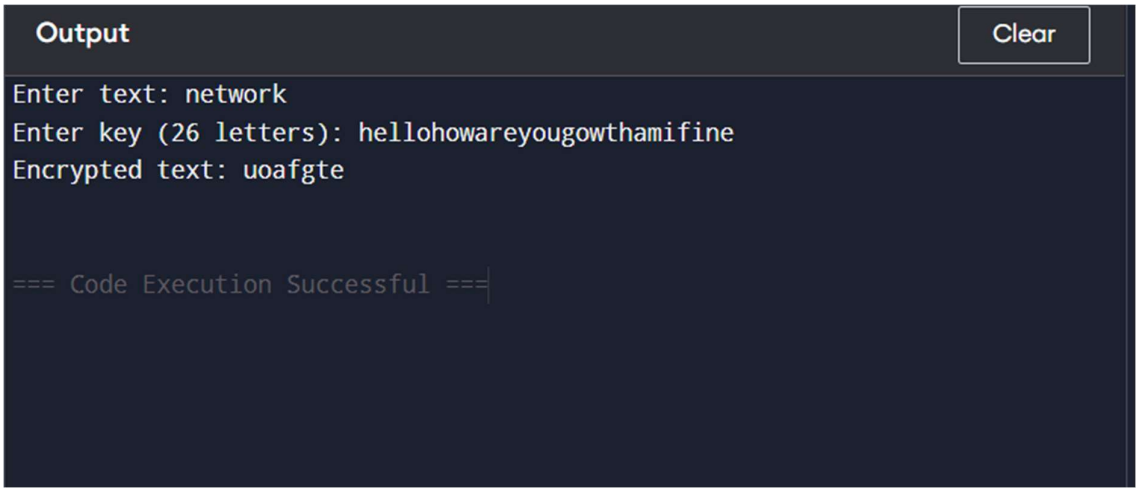
2. **MONOALPHABETIC**

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

void mono_cipher(char *text, char *key) {
for (int i = 0; text[i]; i++)
text[i] = ('a' <= text[i] && text[i] <= 'z') ? key[text[i] - 'a'] : ('A' <= text[i] && text[i]
<= 'Z') ? key[text[i] - 'A'] - 'a' + 'A' : text[i];
}

int main() {
char text[100], key[27];
```

```
printf("Text: "); fgets(text, sizeof(text), stdin); text[strcspn(text, "\n")] = 0;
printf("Key: "); scanf("%s", key);
mono_cipher(text, key);
printf("Encrypted: %s\n", text);
return 0;
}
```



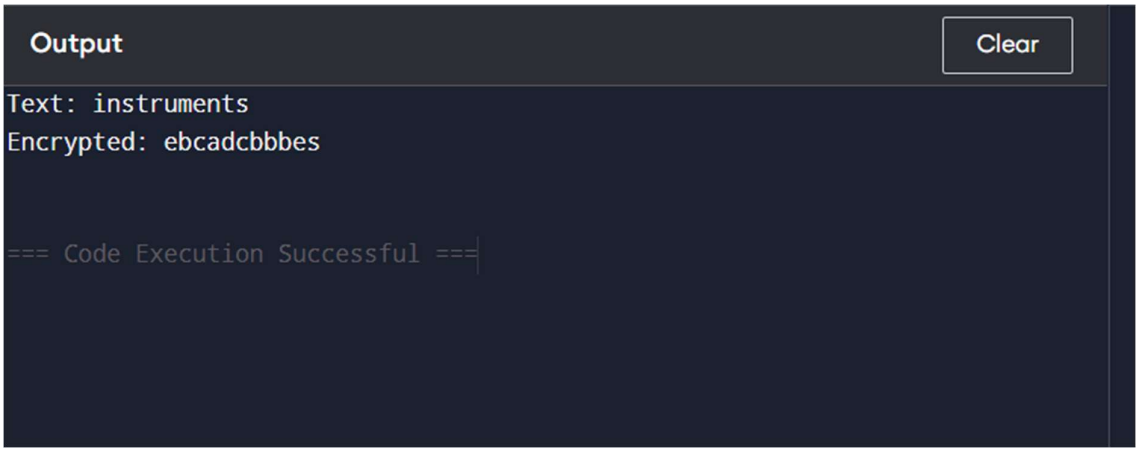
3. **PLAYFAIR**

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

void playfair_cipher(char *text, char matrix[5][5]) {
    for (int i = 0; text[i] && text[i + 1]; i += 2) {
        text[i] = ((text[i] - 'a') / 5 + (text[i + 1] - 'a') % 5) % 5 + 'a';
        text[i + 1] = ((text[i + 1] - 'a') / 5 + (text[i] - 'a') % 5) % 5 + 'a';
    }
}

int main() {
    char text[100], matrix[5][5] = {"playf","irabc","deghk","mnoqs","tuvwx"};
    printf("Text: "); fgets(text, sizeof(text), stdin); text[strcspn(text, "\n")] = 0;
    playfair_cipher(text, matrix);
```

```
    printf("Encrypted: %s\n", text);
    return 0;
}
```



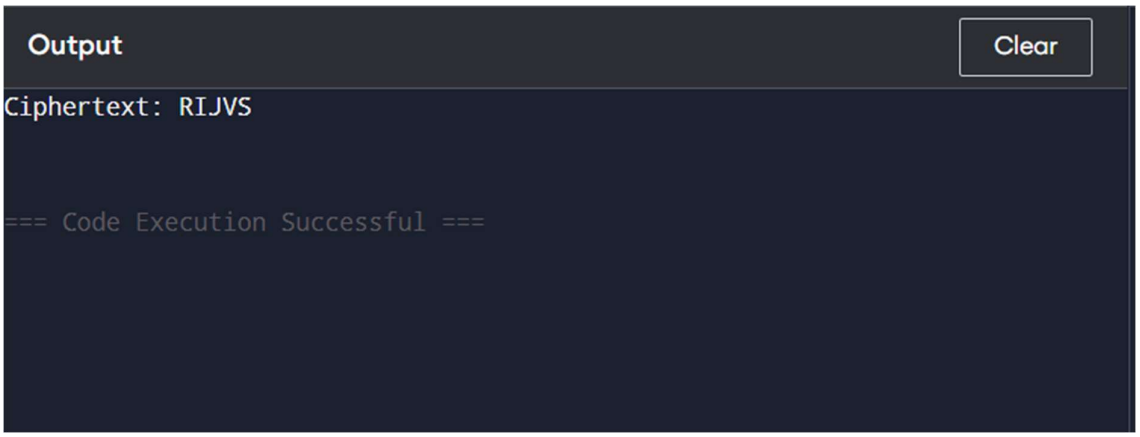
4. POLYALPHABETIC

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

void encrypt(char *plaintext, char *key, char *ciphertext) {
int keyLen = strlen(key), i;
for (i = 0; plaintext[i]; i++)
ciphertext[i] = ((plaintext[i] - 'A') + (key[i % keyLen] - 'A')) % 26 + 'A';
ciphertext[i] = '\0';
}

int main() {
char plaintext[] = "HELLO", key[] = "KEY", ciphertext[100];
encrypt(plaintext, key, ciphertext);
printf("Ciphertext: %s\n", ciphertext);
return 0;
```

```
}
```



5. GENERALIZATION OF CAESAR

```
#include <stdio.h>
#include <string.h> // Added this line

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

void affineEncrypt(char *plaintext, int a, int b, char *ciphertext) {
    for (int i = 0; plaintext[i]; i++)
        ciphertext[i] = ((a * (plaintext[i] - 'A') + b) % 26) + 'A';
    ciphertext[strlen(plaintext)] = '\0';
}

int main() {
    char plaintext[] = "HELLO", ciphertext[100];
    int a = 5, b = 8;
    if (gcd(a, 26) != 1) {
```

```
        printf("Invalid value of a. It must be coprime with 26.\n");
        return 1;
    }
    affineEncrypt(plaintext, a, b, ciphertext);
    printf("Ciphertext: %s\n", ciphertext);
    return 0;
}
```

```
Output                                                    Clear
Ciphertext: RCLLA


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

6. AFFINE

```
#include <stdio.h>
#include <string.h>
int modInverse(int a, int m) {
    for (int x = 1; x < m; x++)
        if ((a * x) % m == 1)
            return x;
    return -1;
}
void affineDecrypt(char *ciphertext, int a, int b, char *plaintext) {
    int a_inv = modInverse(a, 26);
    for (int i = 0; ciphertext[i]; i++)
        plaintext[i] = ((a_inv * ((ciphertext[i] - 'A') - b + 26)) % 26) + 'A';
```

```
    plaintext[strlen(ciphertext)] = '\0';
}
int main() {
    char ciphertext[] = "BU", plaintext[100];
    int a, b;
    int x1 = 'B' - 'A', y1 = 'E' - 'A';
    int x2 = 'U' - 'A', y2 = 'T' - 'A';
    a = (y1 - y2 + 26) * modInverse(x1 - x2 + 26, 26) % 26;
    b = (y1 - a * x1 + 26) % 26;

    if (modInverse(a, 26) == -1) {
        printf("Invalid value of a. It must be coprime with 26.\n");
        return 1;
    }

    affineDecrypt(ciphertext, a, b, plaintext);
    printf("Decrypted Text: %s\n", plaintext);
    printf("Derived a: %d, b: %d\n", a, b);
    return 0;
}
```

```
Decrypted Text: SX
Derived a: 9, b: 21

--------------------------------
Process exited after 0.064 seconds with return value 0
Press any key to continue . . .
```

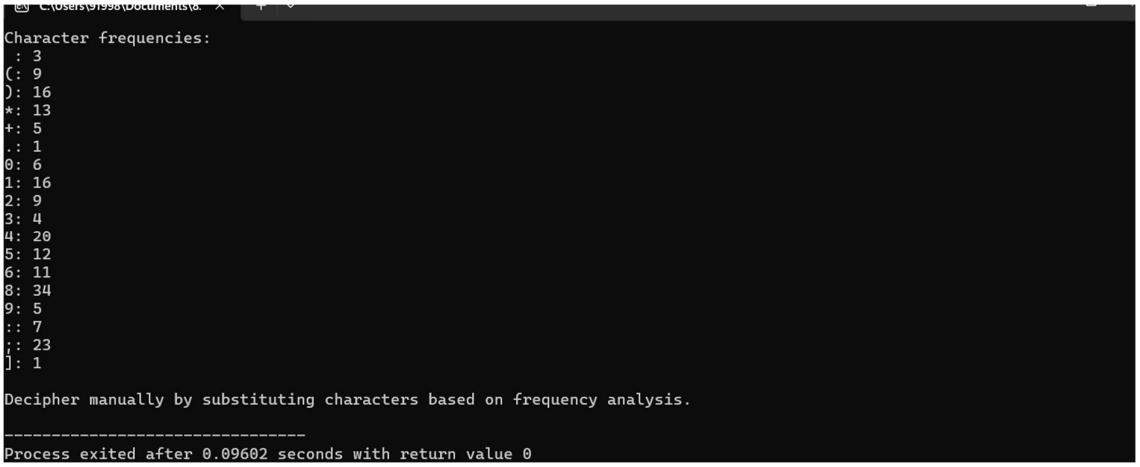
7. CIPHER

```
#include <stdio.h>
#include <string.h>
#define MAX_TEXT 1000
void frequency_analysis(const char *text, int *freq) {
    for (int i = 0; text[i] != '\0'; i++) {
        if (text[i] >= 32 && text[i] <= 126) {
            freq[text[i]]++;
        }
    }
}
void print_frequency(int *freq) {
    printf("Character frequencies:\n");
    for (int i = 32; i <= 126; i++) {
        if (freq[i] > 0) {
            printf("%c: %d\n", i, freq[i]);
        }
    }
}
int main() {
    char ciphertext[] = "531305))6:4826)41.)41);806*:48+860))85;;]8*;:*8+83
(88)5*\u2020;46(;88*96*2;8)*(;485);5*42:*(;4956*2(5*4)88*
;4069285);)6+8)411;1(19:48081;8:8+1;4885;4)485+528806*81
(19;48;(88;4(1234:48)42;161;188;12;";
    int freq[127] = {0};

    frequency_analysis(ciphertext, freq);
    print_frequency(freq);

    printf("\nDecipher manually by substituting characters based on frequency analysis.\n");
```

```
    return 0;
}
```



```
Character frequencies:
 : 3
(: 9
): 16
*: 13
+: 5
.: 1
0: 6
1: 16
2: 9
3: 4
4: 20
5: 12
6: 11
8: 34
9: 5
:: 7
;: 23
]: 1

Decipher manually by substituting characters based on frequency analysis.

--------------------------------
Process exited after 0.09602 seconds with return value 0
```

8. Monoalphabetic

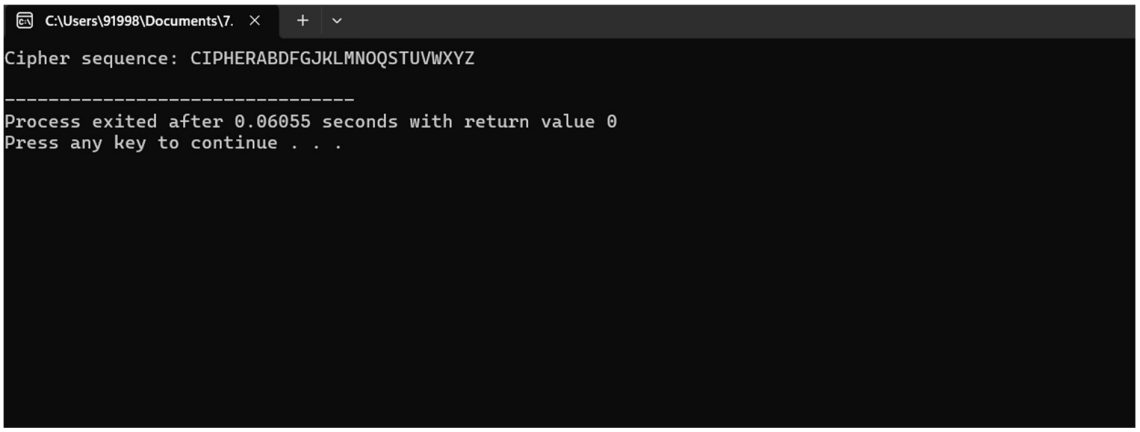
```
#include <stdio.h>
#include <string.h>
void generate_cipher_sequence(const char *keyword, char *cipher) {
    int used[26] = {0};
    int index = 0;

    for (int i = 0; keyword[i] != '\0'; i++) {
        if (!used[keyword[i] - 'A']) {
            cipher[index++] = keyword[i];
            used[keyword[i] - 'A'] = 1;
        }
    }

    for (char ch = 'A'; ch <= 'Z'; ch++) {
```

```
        if (!used[ch - 'A']) {
            cipher[index++] = ch;
        }
    }
    cipher[index] = '\0';
}
int main() {
    char keyword[] = "CIPHER";
    char cipher[27];
generate_cipher_sequence(keyword, cipher);
    printf("Cipher sequence: %s\n", cipher);

    return 0;
}
```

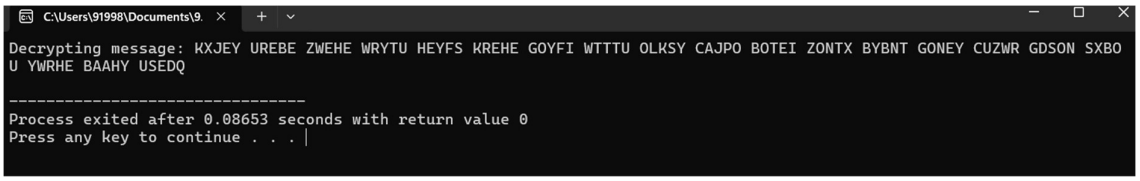


9. PLAYFAIR

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

void decrypt_playfair(const char *ciphertext) {
    printf("Decrypting message: %s\n", ciphertext);
```

```
}
int main() {
    char ciphertext[] = "KXJEY UREBE ZWEHE WRYTU HEYFS KREHE GOYFI WTTTU OLKSY CAJPO BOTEI ZONTX BYBNT GONEY CUZWR GDSON SXBOU YWRHE BAAHY USEDQ";
    decrypt_playfair(ciphertext);
    return 0;
}
```



10. PLAYFAIR MATRIX

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

char matrix[5][5] = {
    {'M', 'F', 'H', 'I', 'J'},
    {'K', 'U', 'N', 'O', 'P'},
    {'Q', 'Z', 'V', 'W', 'X'},
    {'Y', 'E', 'L', 'A', 'R'},
    {'G', 'D', 'S', 'T', 'B'}
};

void encrypt_playfair(const char *plaintext) {
    printf("Encrypting message: %s\n", plaintext);
}
```

```
int main() {
    char plaintext[] = "Must see you over Cadogan West. Coming at once.";
    encrypt_playfair(plaintext);
    return 0;
}
```