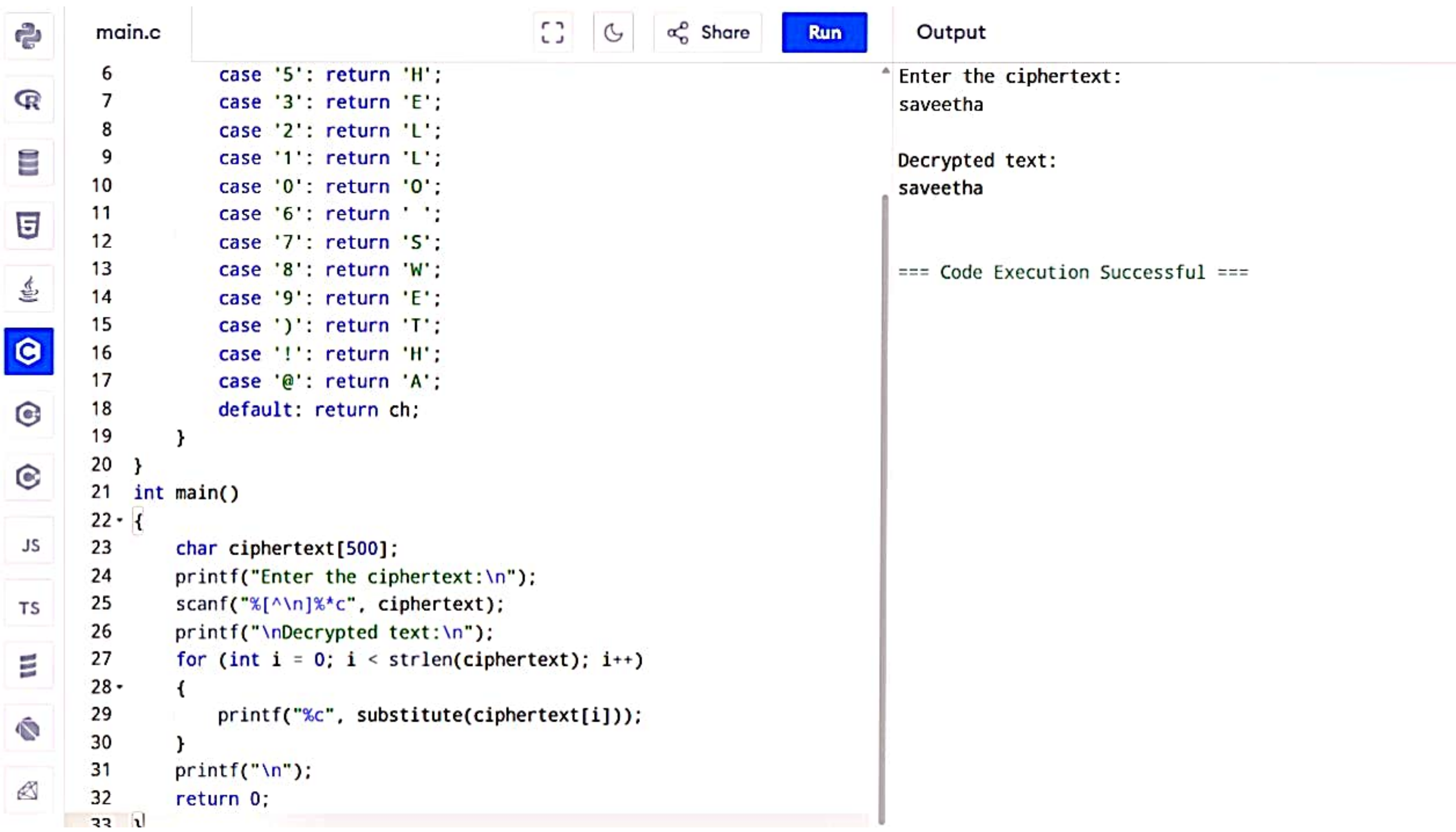
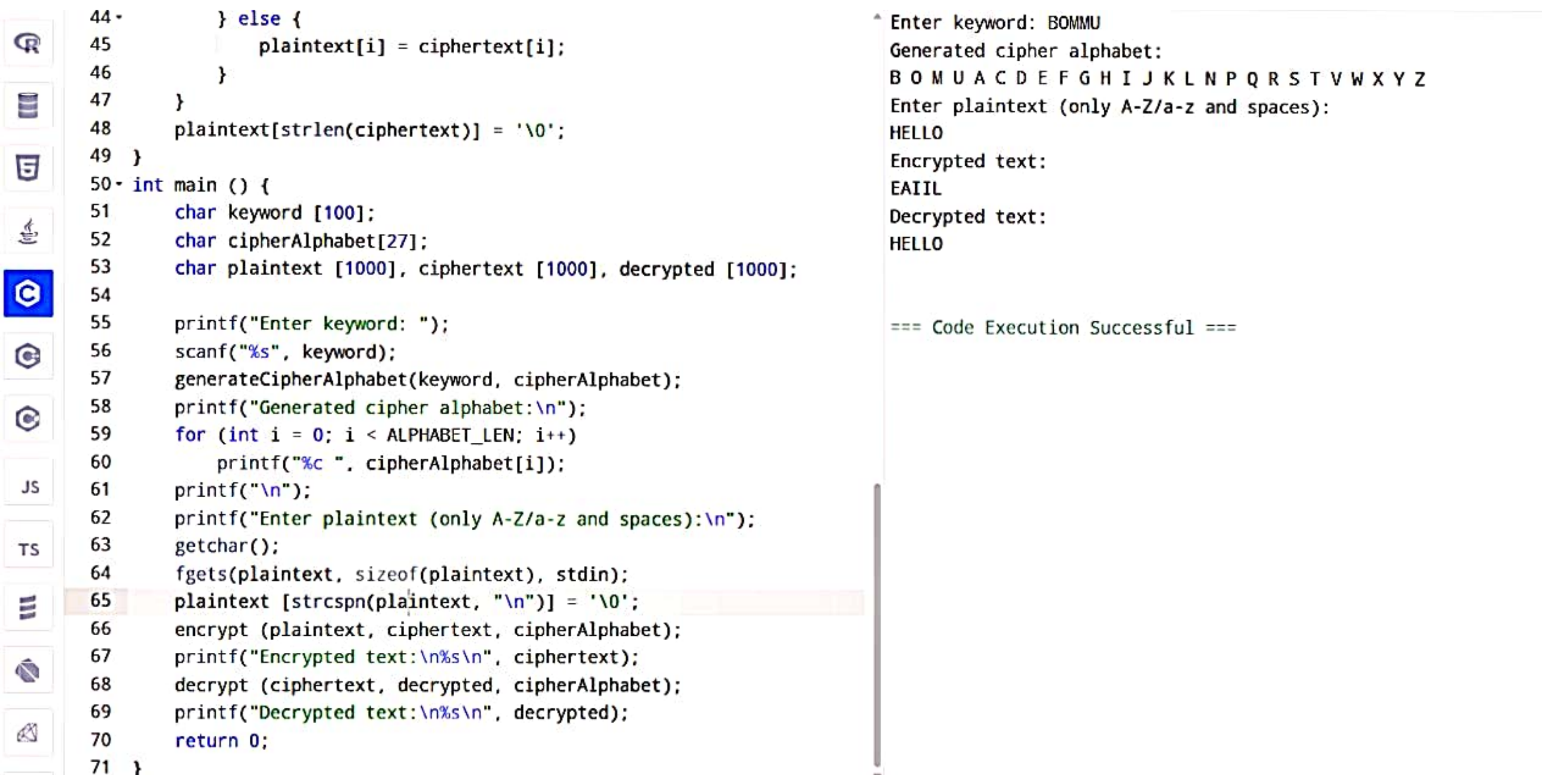


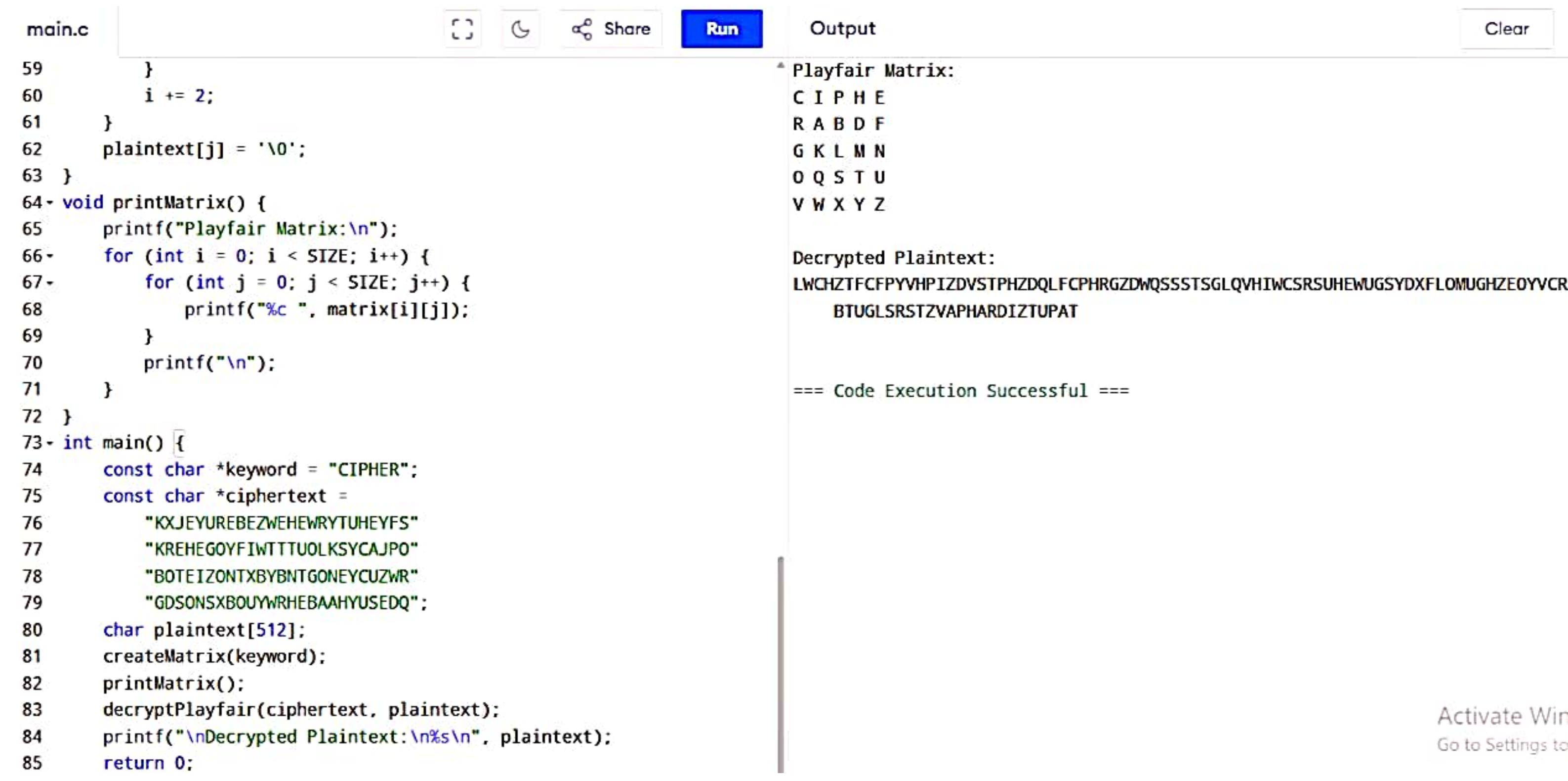
```c
        case '5': return 'H';
        case '3': return 'E';
        case '2': return 'L';
        case '1': return 'L';
        case '0': return 'O';
        case '6': return ' ';
        case '7': return 'S';
        case '8': return 'W';
        case '9': return 'E';
        case ')': return 'T';
        case '!': return 'H';
        case '@': return 'A';
        default: return ch;
    }
}
int main()
{
    char ciphertext[500];
    printf("Enter the ciphertext:\n");
    scanf("%[^\n]%*c", ciphertext);
    printf("\nDecrypted text:\n");
    for (int i = 0; i < strlen(ciphertext); i++)
    {
        printf("%c", substitute(ciphertext[i]));
    }
    printf("\n");
    return 0;
}
```

```
Enter the ciphertext:
saveetha

Decrypted text:
saveetha


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



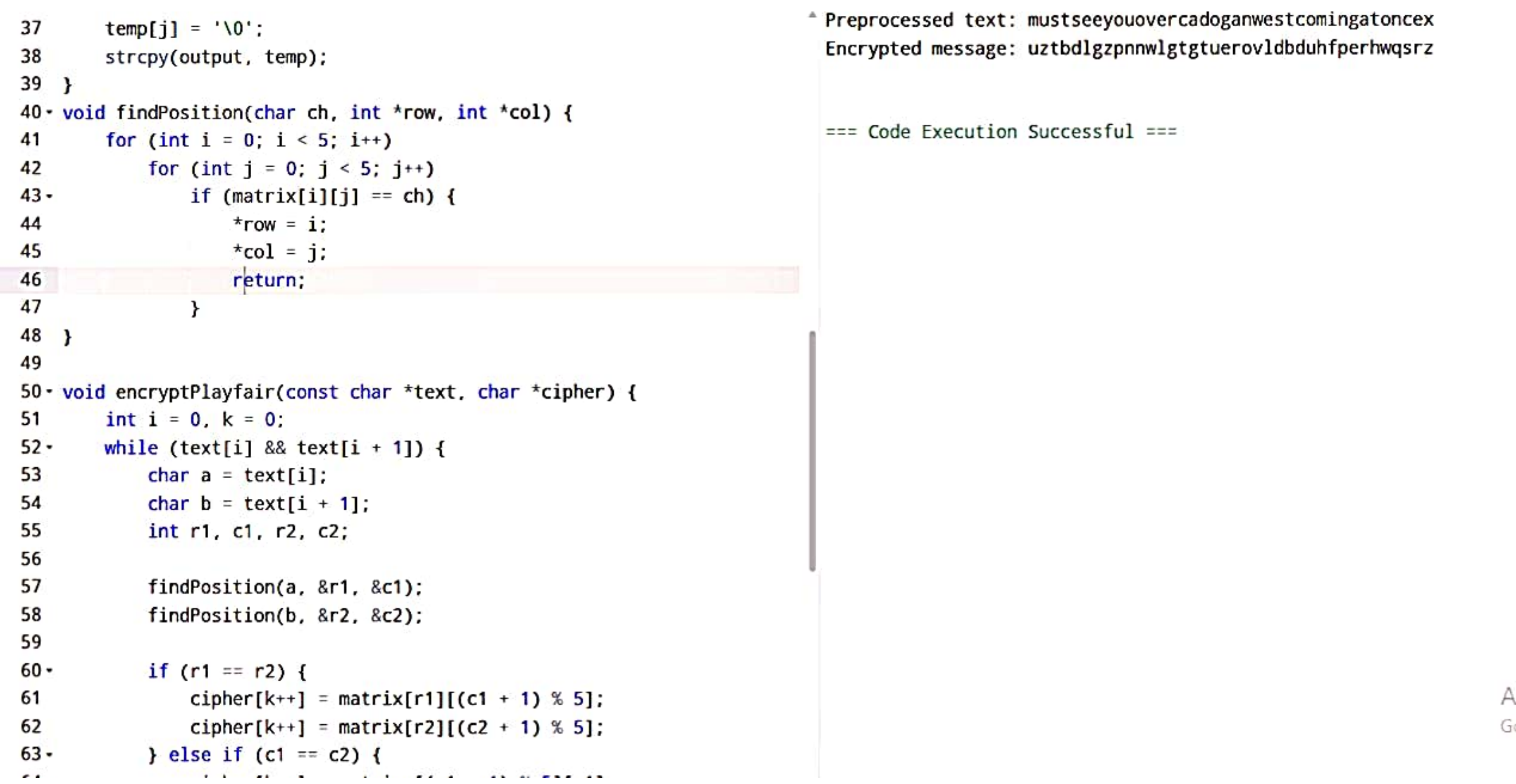
```c
        } else {
            plaintext[i] = ciphertext[i];
        }
    }
    plaintext[strlen(ciphertext)] = '\0';
}
int main () {
    char keyword [100];
    char cipherAlphabet[27];
    char plaintext [1000], ciphertext [1000], decrypted [1000];

    printf("Enter keyword: ");
    scanf("%s", keyword);
    generateCipherAlphabet(keyword, cipherAlphabet);
    printf("Generated cipher alphabet:\n");
    for (int i = 0; i < ALPHABET_LEN; i++)
        printf("%c ", cipherAlphabet[i]);
    printf("\n");
    printf("Enter plaintext (only A-Z/a-z and spaces):\n");
    getchar();
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext [strcspn(plaintext, "\n")] = '\0';
    encrypt (plaintext, ciphertext, cipherAlphabet);
    printf("Encrypted text:\n%s\n", ciphertext);
    decrypt (ciphertext, decrypted, cipherAlphabet);
    printf("Decrypted text:\n%s\n", decrypted);
    return 0;
}
```

```
Enter keyword: BOMMU
Generated cipher alphabet:
B O M U A C D E F G H I J K L N P Q R S T V W X Y Z
Enter plaintext (only A-Z/a-z and spaces):
HELLO
Encrypted text:
EAIIL
Decrypted text:
HELLO


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



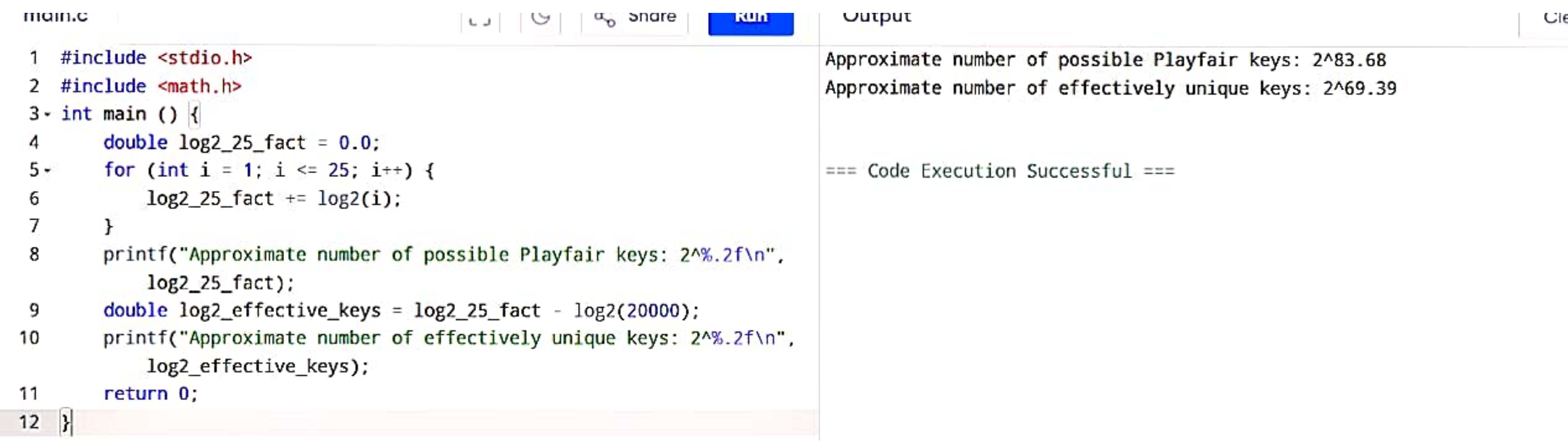
```c
        }
        i += 2;
    }
    plaintext[j] = '\0';
}
void printMatrix() {
    printf("Playfair Matrix:\n");
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            printf("%c ", matrix[i][j]);
        }
        printf("\n");
    }
}
int main() {
    const char *keyword = "CIPHER";
    const char *ciphertext =
        "KXJEYUREBEZWEHEWRYTUHEYFS"
        "KREHEGOYFIWTTTUOLKSYCAJPO"
        "BOTEIZONTXBYBNTGONEYCUZWR"
        "GDSONSXBOUYWRHEBAAHYUSEDQ";
    char plaintext[512];
    createMatrix(keyword);
    printMatrix();
    decryptPlayfair(ciphertext, plaintext);
    printf("\nDecrypted Plaintext:\n%s\n", plaintext);
    return 0;
```

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

Decrypted Plaintext:
LWCHZTFCFPYVHPIZDVSTPHZDQLFCPHRGZDWQSSSTSGLQVHIWCSRSUHEWUGSYDXFLOMUGHZEOYVCR
    BTUGLSRSTZVAPHARDIZTUPAT


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



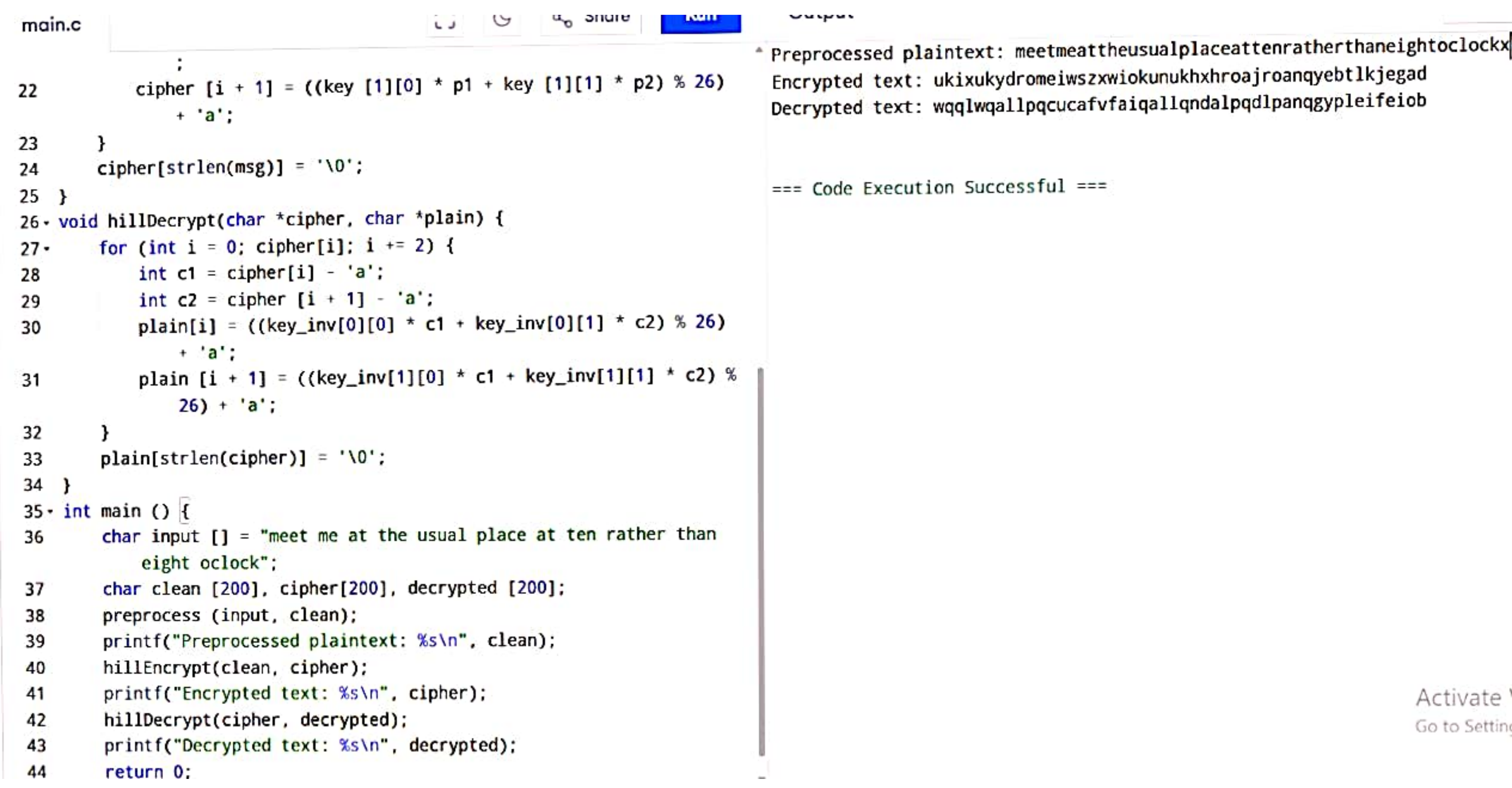
```c
    temp[j] = '\0';
    strcpy(output, temp);
}
void findPosition(char ch, int *row, int *col) {
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 5; j++)
            if (matrix[i][j] == ch) {
                *row = i;
                *col = j;
                return;
            }
}

void encryptPlayfair(const char *text, char *cipher) {
    int i = 0, k = 0;
    while (text[i] && text[i + 1]) {
        char a = text[i];
        char b = text[i + 1];
        int r1, c1, r2, c2;

        findPosition(a, &r1, &c1);
        findPosition(b, &r2, &c2);

        if (r1 == r2) {
            cipher[k++] = matrix[r1][(c1 + 1) % 5];
            cipher[k++] = matrix[r2][(c2 + 1) % 5];
        } else if (c1 == c2) {
```

```
Preprocessed text: mustseeyouovercadoganwestcomingatoncex
Encrypted message: uztbdlgzpnnwlgtgtuerovldbduhfperhwqsrz


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



```c
#include <stdio.h>
#include <math.h>
int main () {
    double log2_25_fact = 0.0;
    for (int i = 1; i <= 25; i++) {
        log2_25_fact += log2(i);
    }
    printf("Approximate number of possible Playfair keys: 2^%.2f\n",
        log2_25_fact);
    double log2_effective_keys = log2_25_fact - log2(20000);
    printf("Approximate number of effectively unique keys: 2^%.2f\n",
        log2_effective_keys);
    return 0;
}
```

```
Approximate number of possible Playfair keys: 2^83.68
Approximate number of effectively unique keys: 2^69.39


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



```c
            ;
        cipher [i + 1] = ((key [1][0] * p1 + key [1][1] * p2) % 26)
            + 'a';
    }
    cipher[strlen(msg)] = '\0';
}
void hillDecrypt(char *cipher, char *plain) {
    for (int i = 0; cipher[i]; i += 2) {
        int c1 = cipher[i] - 'a';
        int c2 = cipher [i + 1] - 'a';
        plain[i] = ((key_inv[0][0] * c1 + key_inv[0][1] * c2) % 26)
            + 'a';
        plain [i + 1] = ((key_inv[1][0] * c1 + key_inv[1][1] * c2) %
            26) + 'a';
    }
    plain[strlen(cipher)] = '\0';
}
int main () {
    char input [] = "meet me at the usual place at ten rather than
        eight oclock";
    char clean [200], cipher[200], decrypted [200];
    preprocess (input, clean);
    printf("Preprocessed plaintext: %s\n", clean);
    hillEncrypt(clean, cipher);
    printf("Encrypted text: %s\n", cipher);
    hillDecrypt(cipher, decrypted);
    printf("Decrypted text: %s\n", decrypted);
    return 0;
```

```
Preprocessed plaintext: meetmeattheusualplaceattenratherthaneightoclockx
Encrypted text: ukixukydromeiwszxwiokunukhxhroajroanqyebtlkjegad
Decrypted text: wqqlwqallpqcucafvfaiqallqndalpqdlpanqgypleifeiob


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