main.c

Share Run

```
        } else if (islower(ch)) {
            ciphertext[i] = ((ch - 'a' + key) % 26) + 'a';
        } else {
            ciphertext[i] = ch;
        }
    }
    ciphertext[i] = '\0';
}
int main () {
    char plaintext [MAX_LEN];
    char ciphertext [MAX_LEN];
    int key;
    printf("Enter a message to encrypt: ");
    fgets(plaintext, MAX_LEN, stdin);
    size_t len = strlen(plaintext);
    if (len > 0 && plaintext [len - 1] == '\n') {
        plaintext [len - 1] = '\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;
    }
    encrypt (plaintext, ciphertext, key);
    printf("Encrypted message: %s\n", ciphertext);
```

Output

```
Enter a message to encrypt: hello world
Enter key (1-25): 5
Encrypted message: mjqqt btwqi


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

main.c

Share Run

```c
    for (int i = 0; i < 26; i++) {
        if (!isalpha(key[i]))
            return 0;
        int index = toupper(key[i]) - 'A';
        if (freq[index]++)
            return 0;
    }
    return 1;
}
int main () {
    char plaintext [MAX_LEN], ciphertext [MAX_LEN];
    char key [27];
    printf("Enter the plaintext: ");
    fgets(plaintext, MAX_LEN, stdin);
    size_t len = strlen(plaintext);
    if (len > 0 && plaintext [len - 1] == '\n')
        plaintext [len - 1] = '\0';
    printf("Enter 26-letter substitution key (A-Z): ");
    fgets(key, 27, stdin);
    if (!isValidKey(key)) {
        printf("Invalid key! Key must be 26 unique alphabetic
            letters.\n");
        return 1;
    }
    encrypt (plaintext, ciphertext, key);
    printf("Encrypted message: %s\n", ciphertext);
```

Output Clear

```
Enter the plaintext: hello world
Enter 26-letter substitution key (A-Z): efghijklmnopqrstuvwxyzabcd
Encrypted message: lipps asvph


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

main.c

Share Run

Output

```
    }
}
for (i = 0; i < 26; i++) {
    if (i + 'a' == 'j' || used[i]) continue;
    m[k / 5][k % 5] = i + 'a';
    k++;
}
printf("Encrypted: ");
for (i = 0; text[i]; i += 2) {
    char a = text[i], b = text[i + 1] ? text[i + 1] : 'x';
    if (a == 'j') a = 'i'; if (b == 'j') b = 'i';
    if (a == b) b = 'x';

    for (j = 0; j < 5; j++)
        for (k = 0; k < 5; k++) {
            if (m[j][k] == a) { r1 = j; c1 = k; }
            if (m[j][k] == b) { r2 = j; c2 = k; }
        }

    if (r1 == r2)
        printf("%c%c", m[r1][(c1 + 1) % 5], m[r2][(c2 + 1) % 5]
            );
    else if (c1 == c2)
        printf("%c%c", m[(r1 + 1) % 5][c1], m[(r2 + 1) % 5][c2]
            );
    else
```

```
Key: MONARCHY
Plaintext: BOMMU
Encrypted: HRHRHR

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

main.c

Share Run

```
    for (int i = 0; i < len; i++) {
        char ch = plaintext[i];
        if (isalpha(ch)) {
            int shift = fullKey[i] - 'A';
            if (isupper(ch)) {
                ciphertext[i] = ((ch - 'A' + shift) % 26) + 'A';
            } else {
                ciphertext[i] = ((ch - 'a' + shift) % 26) + 'a';
            }
        } else {
            ciphertext[i] = ch;
        }
    }
    ciphertext[len] = '\0';
}
int main() {
    char plaintext[MAX_LEN], key[MAX_LEN], ciphertext[MAX_LEN];
    printf("Enter the plaintext: ");
    fgets(plaintext, MAX_LEN, stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    printf("Enter the keyword: ");
    fgets(key, MAX_LEN, stdin);
    key[strcspn(key, "\n")] = '\0';
    encrypt(plaintext, key, ciphertext);
    printf("Encrypted text: %s\n", ciphertext);
```

Output

```
Enter the plaintext: sudharshan
Enter the keyword: bommu
Encrypted text: tipturshan


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

main.c

Share Run

```
        printf("Decryption not possible. No modular inverse for a =
            %d.\n", a);
        return;
    }
    for (int i = 0; ciphertext[i] != '\0'; i++) {
        decrypted[i] = decryptChar(ciphertext[i], aInv, b);
    }
    decrypted[strlen(ciphertext)] = '\0';
}
int main() {
    char plaintext[1000], ciphertext[1000], decrypted[1000];
    int a, b;
    printf("Enter plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    printf("Enter values of a and b (a must be coprime with 26): ");
    scanf("%d %d", &a, &b);
    if (gcd(a, 26) != 1) {
        printf("Invalid value of a. It must be coprime with 26.\n");
        return 1;
    }
    encryptText(plaintext, ciphertext, a, b);
    printf("Encrypted text: %s\n", ciphertext);
    decryptText(ciphertext, decrypted, a, b);
    printf("Decrypted text: %s\n", decrypted);
    return 0;
}
```

Output

```
Enter plaintext: hello
Enter values of a and b (a must be coprime with 26): 5 8
Encrypted text: rclla
Decrypted text: hello


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

main.c

Share Run

```c
    int dp = (p1 - p2 + 26) % 26;
    int dc = (c1 - c2 + 26) % 26;
    int inv = modInverse(dp, 26);
    if (inv == -1)
        return 0;
    *a = (dc * inv) % 26;
    *b = (c1 - (*a * p1) + 26 * 26) % 26;
    return 1;
}
int main() {
    char ciphertext[1000], plaintext[1000];
    int a, b;
    printf("Enter ciphertext (uppercase letters only):\n");
    fgets(ciphertext, sizeof(ciphertext), stdin);
    ciphertext[strcspn(ciphertext, "\n")] = '\0';
    int c1 = 'B' - 'A'; // 1
    int c2 = 'U' - 'A'; // 20
    int p1 = 'E' - 'A'; // 4
    int p2 = 'T' - 'A'; // 19

    if (!solveAffineKeys(c1, p1, c2, p2, &a, &b)) {
        printf("Failed to solve affine key equations.\n");
        return 1;
    }
    printf("Guessed keys: a = %d, b = %d\n", a, b);
    decryptText(ciphertext, plaintext, a, b);
    printf("Decrypted text:\n%s\n", plaintext);
    return 0;
```

Output

```
Enter ciphertext (uppercase letters only):
BOMMU
Guessed keys: a = 3, b = 15
Decrypted text:
ERZZT


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