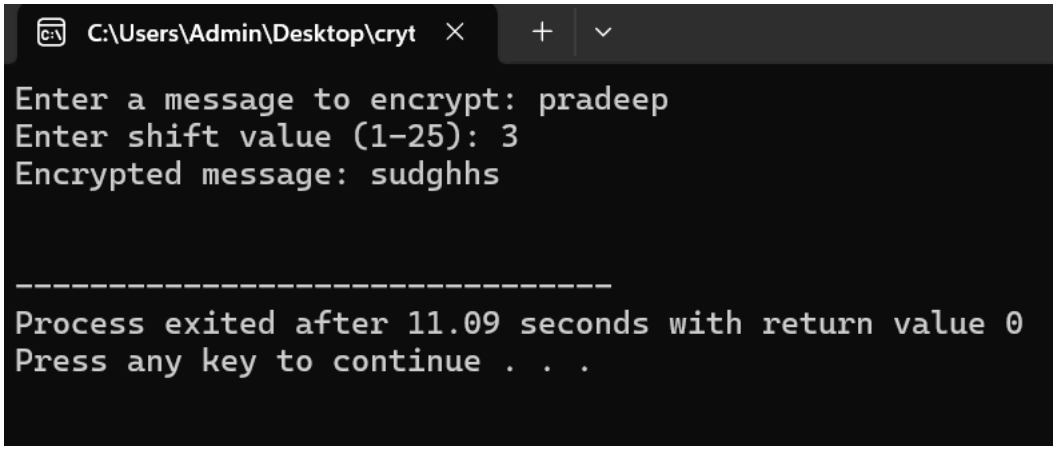
1.write a C program for Caesar cipher involves replacing each letter of the alphabet with the letter standing k places further down the alphabet, for k in the range 1 through 25.

Program:

```
#include <stdio.h>
#include <ctype.h>
void encrypt(char text[], int shift) {
    for (int i = 0; text[i] != '\0'; ++i) {
        char ch = text[i];
        if (isalpha(ch)) {
            if (isupper(ch)) {
                text[i] = ((ch - 'A' + shift) % 26) + 'A';
            } else {
                text[i] = ((ch - 'a' + shift) % 26) + 'a';
            }
        }
    }
}
int main() {
    char text[100];
    int shift;
    printf("Enter a message to encrypt: ");
    fgets(text, sizeof(text), stdin);
    printf("Enter shift value (1-25): ");
    scanf("%d", &shift);
    if (shift < 1 || shift > 25) {
        printf("Invalid shift value. Please enter a number between 1 and 25.\n");
        return 1;
    }encrypt(text, shift);
    printf("Encrypted message: %s\n", text);
    return 0;
}
```

Sample output:




2. Write a C program for monoalphabetic substitution cipher maps a plaintext alphabet to a ciphertext alphabet, so that each letter of the plaintext alphabet maps to a single unique letter of the ciphertext alphabet.

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
char plaintext_alphabet[] = "abcdefghijklmnopqrstuvwxyz";
char ciphertext_alphabet[] = "QWERTYUIOPASDFGHJKLZXCVBNM";
void encrypt(char text[]) {
    for (int i = 0; text[i] != '\0'; ++i) {
        if (isalpha(text[i])) {
            char ch = tolower(text[i]);
            int index = ch - 'a';
            if (islower(text[i])) {
                text[i] = tolower(ciphertext_alphabet[index]);
            } else {
                text[i] = toupper(ciphertext_alphabet[index]);
            }
        }
    }
}
```

```
int main() {
    char text[100];
    printf("Enter a message to encrypt: ");
    fgets(text, sizeof(text), stdin);
    encrypt(text);
    printf("Encrypted message: %s\n", text);
    return 0;
}
```

Sample output:

```
Enter a message to encrypt: pradeep
Encrypted message: hkqrtth


--------------------------------
Process exited after 6.606 seconds with return value 0
Press any key to continue . . .
```

3. Write a C program for Playfair algorithm is based on the use of a 5 X 5 matrix of letters constructed using a keyword. Plaintext is encrypted two letters at a time using this matrix.

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 5
void removeDuplicates(char *str) {
    int length = strlen(str);
    for (int i = 0; i < length; i++) {
        for (int j = i + 1; j < length; j++) {
            if (str[i] == str[j]) {
                for (int k = j; k < length; k++) {
                    str[k] = str[k + 1];
                }
```

```
            length--;
            j--;
        }
    }
}
void createMatrix(char *key, char matrix[SIZE][SIZE]) {
    int k = 0;
    int used[26] = {0};
    for (int i = 0; key[i] != '\0'; i++) {
        char c = toupper(key[i]);
        if (c >= 'A' && c <= 'Z' && !used[c - 'A']) {
            matrix[k / SIZE][k % SIZE] = c;
            used[c - 'A'] = 1;
            k++;
        }
    }
    for (char c = 'A'; c <= 'Z'; c++) {
        if (c != 'J' && !used[c - 'A']) {
            matrix[k / SIZE][k % SIZE] = c;
            k++;
        }
    }
}
void preprocessPlaintext(char *text, char *processedText) {
    int k = 0;
    for (int i = 0; text[i] != '\0'; i++) {
        if (isalpha(text[i])) {
            processedText[k++] = toupper(text[i]);
        }
```

```
    }
    for (int i = 0; i < k - 1; i += 2) {
        if (processedText[i] == processedText[i + 1]) {
            for (int j = k; j > i + 1; j--) {
                processedText[j] = processedText[j - 1];
            }
            processedText[i + 1] = 'X';
            k++;
        }
    }
    if (k % 2 != 0) {
        processedText[k++] = 'X';
    }
    processedText[k] = '\0';
}
void findPosition(char c, char matrix[SIZE][SIZE], int *row, int *col) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (matrix[i][j] == c) {
                *row = i;
                *col = j;
                return;
            }
        }
    }
}
void encryptPlayfair(char *text, char matrix[SIZE][SIZE], char *ciphertext) {
    int k = 0;
    for (int i = 0; text[i] != '\0'; i += 2) {
        int row1, col1, row2, col2;
```

```
        findPosition(text[i], matrix, &row1, &col1);
        findPosition(text[i + 1], matrix, &row2, &col2);

        if (row1 == row2) {
            ciphertext[k++] = matrix[row1][(col1 + 1) % SIZE];
            ciphertext[k++] = matrix[row2][(col2 + 1) % SIZE];
        } else if (col1 == col2) {
            ciphertext[k++] = matrix[(row1 + 1) % SIZE][col1];
            ciphertext[k++] = matrix[(row2 + 1) % SIZE][col2];
        } else {
            ciphertext[k++] = matrix[row1][col2];
            ciphertext[k++] = matrix[row2][col1];
        }
    }
    ciphertext[k] = '\0';
}

int main() {
    char key[100], plaintext[100], processedText[200], ciphertext[200];
    char matrix[SIZE][SIZE];
    printf("Enter the keyword: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = '\0';
    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    removeDuplicates(key);
    createMatrix(key, matrix);
    preprocessPlaintext(plaintext, processedText);
    encryptPlayfair(processedText, matrix, ciphertext);
```

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

Sample output:

```
Enter the keyword: key
Enter the plaintext: pradeep chem
Encrypted message: QSEGAVKQDCYL

--------------------------------
Process exited after 21.09 seconds with return value 0
Press any key to continue . . .
```

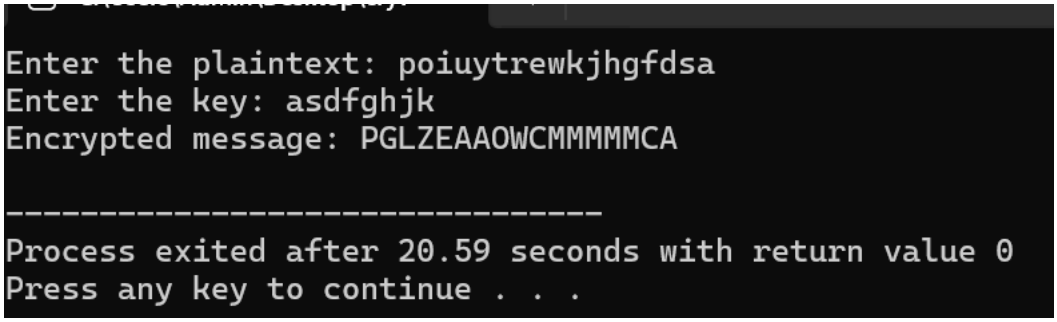
4. Write a C program for polyalphabetic substitution cipher uses a separate monoalphabetic substitution cipher for each successive letter of plaintext, depending on a key.

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void polyalphabeticCipher(char *plaintext, char *key, char *ciphertext) {
    int i, j = 0;
    int len = strlen(plaintext);
    for (i = 0; i < len; i++) {
        if (isalpha(plaintext[i])) {
            char p = toupper(plaintext[i]);
            char k = toupper(key[j % strlen(key)]);
            ciphertext[i] = ((p - 'A' + (k - 'A')) % 26) + 'A';
            j++;
        } else {
            ciphertext[i] = plaintext[i];
        }
    }
    ciphertext[i] = '\0';
}
```

```
int main() {
    char plaintext[100], key[100], ciphertext[100];
    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    printf("Enter the key: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = '\0';
    polyalphabeticCipher(plaintext, key, ciphertext);
    printf("Encrypted message: %s\n", ciphertext);
    return 0;
}
```

Sample output:




**5. Write a C program for generalization of the Caesar cipher, known as the affine Caesar cipher, has the following form: For each plaintext letter p, substitute the ciphertext letter C: C = E([a, b], p) = (ap + b) mod 26 A basic requirement of any encryption algorithm is that it be one-to-one. That is, if p q, then E(k, p) E(k, q). Otherwise, decryption is impossible, because more than one plaintext character maps into the same ciphertext character. The affine Caesar cipher is not one-to-one for all values of a. For example, for a = 2 and b = 3, then E([a, b], 0) = E([a, b], 13) = 3. a. Are there any limitations on the value of b? b. Determine which values of a are not allowed.**

**Program:**

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int gcd(int a, int b) {
    while (b != 0) {
```

```
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
void affineCipher(char *plaintext, int a, int b, char *ciphertext) {
    int i;
    for (i = 0; i < strlen(plaintext); i++) {
        if (isalpha(plaintext[i])) {
            char p = toupper(plaintext[i]);
            int p_num = p - 'A';

            int c_num = (a * p_num + b) % 26;

            ciphertext[i] = c_num + 'A';
        } else {
            ciphertext[i] = plaintext[i];
        }
    }
    ciphertext[i] = '\0';
}
int main() {
    char plaintext[100], ciphertext[100];
    int a, b;
    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0';
    printf("Enter the key a (should be coprime with 26): ");
    scanf("%d", &a);
```

```
    printf("Enter the key b: ");
    scanf("%d", &b);
    if (gcd(a, 26) != 1) {
        printf("Error: 'a' must be coprime with 26.\n");
        return 1;
    }
    affineCipher(plaintext, a, b, ciphertext);
    printf("Encrypted message: %s\n", ciphertext);

    return 0;
}
```

**Sa**mple output:

```
Enter the plaintext: lkjhgfdsamnbvcxz
Enter the key a (should be coprime with 26): 7
Enter the key b: 7
Encrypted message: GZSEXQCDHNUOYVMA

--------------------------------
Process exited after 34.39 seconds with return value 0
Press any key to continue . . .
```

6. Write a C program for ciphertext has been generated with an affine cipher. The most frequent letter of the ciphertext is “B,” and the second most frequent letter of the ciphertext is “U.”Break this code.

Program:

```
#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;
}
```

```
char affineDecryption(char c, int a, int b) {
    if (c >= 'A' && c <= 'Z') {
        int x = c - 'A';
        int y = (a * (x - b + 26)) % 26;
        return y + 'A';
    }
    return c;
}
void breakAffineCipher(char *ciphertext) {

    int C1 = 'B' - 'A';
    int C2 = 'U' - 'A';
    int P1 = 'E' - 'A';
    int P2 = 'T' - 'A';
    int a, b;
    int found = 0;
    for (a = 1; a < 26; a++) {
        if (a == 13 || a == 26 - 13)
            continue;
        int a_inv = modInverse(a, 26);
        if (a_inv == -1) continue;
        b = (C1 - a * P1 + 26) % 26;

        if ((a * P2 + b) % 26 == C2) {
            found = 1;
            break;
        }
    }

    if (found) {
```

```
        printf("Found a = %d, b = %d\n", a, b);
        printf("Decrypted message: ");
        for (int i = 0; i < strlen(ciphertext); i++) {
            printf("%c", affineDecryption(ciphertext[i], a, b));
        }
        printf("\n");
    } else {
        printf("Unable to break the cipher.\n");
    }
}

int main() {
    char ciphertext[100];
    printf("Enter the ciphertext: ");
    fgets(ciphertext, sizeof(ciphertext), stdin);
    ciphertext[strcspn(ciphertext, "\n")] = '\0';
    breakAffineCipher(ciphertext);
    return 0;
}
```

Sample output:

```
Enter the ciphertext: lkjhgfdsamnbvcxz
Found a = 3, b = 15
Decrypted message: lkjhgfdsamnbvcxz

--------------------------------
Process exited after 33.45 seconds with return value 0
Press any key to continue . . .
```

7. Write a C program for the following ciphertext was generated using a simple substitution algorithm.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;‡?;

Program:

```
#include <stdio.h>
#include <string.h>
#define MAX_TEXT_LEN 500
void frequencyAnalysis(char *ciphertext) {
    int freq[256] = {0};
    for (int i = 0; ciphertext[i] != '\0'; i++) {
        if (ciphertext[i] >= 32 && ciphertext[i] <= 126) {
            freq[(int)ciphertext[i]]++;
        }
    }
    printf("Character Frequency Analysis:\n");
    for (int i = 32; i <= 126; i++) {
        if (freq[i] > 0) {
            printf("Character %c: %d times\n", i, freq[i]);
        }
    }
}

int main() {
    char ciphertext[] = "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;‡?;";

    printf("Ciphertext:\n%s\n\n", ciphertext);
    frequencyAnalysis(ciphertext);

    return 0;
}
```

Sample output:

```
Ciphertext:
53ççå305))6*;4826)4ç.)4ç);806*;48å8‖60))85;;]8*;:ç*8å83 (88)5*å;46(;88*96*?;8)*ç(;485);5*å2:*ç(;4956*2(5*ù4)8‖8* ;406928
5);)6å8)4çç;1(ç9;48081;8:8ç1;48å85;4)485å528806*81 (ç9;48;(88;4(ç?34;48)4ç;161;:188;ç?;

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

8. Write a C program for monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. A common technique for avoiding this is to use a keyword from which the cipher sequence can be generated.

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define ALPHABET_LEN 26
void createCipherAlphabet(char cipherAlphabet[], const char keyword[]) {
    int used[ALPHABET_LEN] = {0};
    int index = 0;
    for (int i = 0; keyword[i] != '\0'; i++) {
        char letter = toupper(keyword[i]);
        if (!used[letter - 'A']) {
            cipherAlphabet[index++] = letter;
            used[letter - 'A'] = 1;
        }
    }
    for (int i = 0; i < ALPHABET_LEN; i++) {
        if (!used[i]) {
            cipherAlphabet[index++] = 'A' + i;
        }
```

```
    }
}
void encrypt(char message[], const char cipherAlphabet[]) {
    for (int i = 0; message[i] != '\0'; i++) {
        if (isalpha(message[i])) {
            int pos = toupper(message[i]) - 'A';
            message[i] = isupper(message[i]) ? cipherAlphabet[pos] : tolower(cipherAlphabet[pos]);
        }
    }
}
void decrypt(char message[], const char cipherAlphabet[]) {
    for (int i = 0; message[i] != '\0'; i++) {
        if (isalpha(message[i])) {
            for (int j = 0; j < ALPHABET_LEN; j++) {
                if (toupper(message[i]) == cipherAlphabet[j]) {
                    message[i] = isupper(message[i]) ? 'A' + j : tolower('A' + j);
                    break;
                }
            }
        }
    }
}

int main() {
    char keyword[] = "CIPHER";
    char cipherAlphabet[ALPHABET_LEN];
    char message[] = "Meet me at the usual place at ten rather than eight oclock";
    createCipherAlphabet(cipherAlphabet, keyword);

    printf("Cipher Alphabet:\n");
```

```
    for (int i = 0; i < ALPHABET_LEN; i++) {
        printf("%c ", cipherAlphabet[i]);
    }
    printf("\n\nOriginal Message:\n%s\n", message);
    encrypt(message, cipherAlphabet);
    printf("\nEncrypted Message:\n%s\n", message);
    decrypt(message, cipherAlphabet);
    printf("\nDecrypted Message:\n%s\n", message);

    return 0;
}
```

Sample output:

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

Original Message:
Meet me at the usual place at ten rather than eight oclock

Encrypted Message:
Keet ke ct tbe usucj njcpe ct tel qctbeq tbcl edabt mpjmpg

Decrypted Message:
Meet me at the usual place at ten rather than eight oclock
```

9. Write a C program for PT-109 American patrol boat, under the command of Lieutenant John F. Kennedy, was sunk by a Japanese destroyer, a message was received at an Australian wireless station in Playfair code:

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MATRIX_SIZE 5
#define ALPHABET_SIZE 26
void removeDuplicates(char *key) {
    int length = strlen(key);
```

```
    for (int i = 0; i < length; i++) {
        for (int j = i + 1; j < length; j++) {
            if (tolower(key[i]) == tolower(key[j])) {
                for (int k = j; k < length; k++) {
                    key[k] = key[k + 1];
                }
                length--;
                j--;
            }
        }
    }
}
void createMatrix(char *key, char matrix[MATRIX_SIZE][MATRIX_SIZE]) {
    int index = 0;
    char alphabet[ALPHABET_SIZE] = "ABCDEFGHIKLMNOPQRSTUVWXYZ";
    removeDuplicates(key);
    for (int i = 0; key[i] != '\0'; i++) {
        if (isalpha(key[i])) {
            matrix[index / MATRIX_SIZE][index % MATRIX_SIZE] = toupper(key[i]);
            index++;
        }
    }
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        if (strchr(key, alphabet[i]) == NULL) {
            matrix[index / MATRIX_SIZE][index % MATRIX_SIZE] = alphabet[i];
            index++;
        }
    }
}
void decryptPair(char *a, char *b, char matrix[MATRIX_SIZE][MATRIX_SIZE]) {
```

```
    int rowA, colA, rowB, colB;
    for (int i = 0; i < MATRIX_SIZE; i++) {
        for (int j = 0; j < MATRIX_SIZE; j++) {
            if (matrix[i][j] == toupper(*a)) {
                rowA = i;
                colA = j;
            }
            if (matrix[i][j] == toupper(*b)) {
                rowB = i;
                colB = j;
            }
        }
    }
    if (rowA == rowB) {
        *a = matrix[rowA][(colA - 1 + MATRIX_SIZE) % MATRIX_SIZE];
        *b = matrix[rowB][(colB - 1 + MATRIX_SIZE) % MATRIX_SIZE];
    }
    else if (colA == colB) {
        *a = matrix[(rowA - 1 + MATRIX_SIZE) % MATRIX_SIZE][colA];
        *b = matrix[(rowB - 1 + MATRIX_SIZE) % MATRIX_SIZE][colB];
    }
    else {
        *a = matrix[rowA][colB];
        *b = matrix[rowB][colA];
    }
}
void decrypt(char *ciphertext, char matrix[MATRIX_SIZE][MATRIX_SIZE], char *plaintext)
{
    int len = strlen(ciphertext);
    int index = 0;
    char a, b;
```

```
    for (int i = 0; i < len; i += 5) {
        a = ciphertext[i];
        b = ciphertext[i + 1];
        decryptPair(&a, &b, matrix);
        plaintext[index++] = a;
        plaintext[index++] = b;
    }
    plaintext[index] = '\0';
}

int main() {
    char key[] = "PT109";
    char ciphertext[] = "KXJEY UREBE ZWEHE WRYTU HEYFS KREHE GOYFI WTTTU OLKSY CAJPO BOTEI ZONTX BYBNT GONEY CUZWR GDSON SXBOU YWRHE BAAHY USEDQ";
    char matrix[MATRIX_SIZE][MATRIX_SIZE];
    char plaintext[500];
    createMatrix(key, matrix);
    decrypt(ciphertext, matrix, plaintext);
    printf("Decrypted message: %s\n", plaintext);
    return 0;
}
```

Sample output:

```
C:\Users\Admin\Desktop\cryt
Decrypted message: LWNQHQGDWBGWLQMEMDCQMQBRPSCYWYKCKHNUFHBMSRGDCFQG

--------------------------------
Process exited after 2.613 seconds with return value 0
Press any key to continue . . .
```

10. Write a C program for Playfair matrix:

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 C

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 5
char playfairMatrix[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 letter, int *row, int *col) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (playfairMatrix[i][j] == letter || (letter == 'J' && playfairMatrix[i][j] == 'I')) {
                *row = i;
                *col = j;
                return;
            }
        }
    }
}
void encryptPair(char *first, char *second) {
    int row1, col1, row2, col2;
    findPosition(*first, &row1, &col1);
    findPosition(*second, &row2, &col2);
```

```
    if (row1 == row2) {
        // Same row, shift right
        *first = playfairMatrix[row1][(col1 + 1) % SIZE];
        *second = playfairMatrix[row2][(col2 + 1) % SIZE];
    } else if (col1 == col2) {
        *first = playfairMatrix[(row1 + 1) % SIZE][col1];
        *second = playfairMatrix[(row2 + 1) % SIZE][col2];
    } else {
        *first = playfairMatrix[row1][col2];
        *second = playfairMatrix[row2][col1];
    }
}
void prepareText(char text[], char preparedText[]) {
    int length = 0;
    for (int i = 0; text[i] != '\0'; i++) {
        if (isalpha(text[i])) {
            char letter = toupper(text[i]);
            if (letter == 'J') letter = 'I';

            if (length > 0 && preparedText[length - 1] == letter) {
                preparedText[length++] = 'X';
            }
            preparedText[length++] = letter;
        }
    }
    if (length % 2 != 0) {
        preparedText[length++] = 'X';
    }
    preparedText[length] = '\0';
```

```
}

// Encrypt message using Playfair cipher
void encryptMessage(char message[]) {
    char preparedText[100];
    prepareText(message, preparedText);

    printf("Prepared Text: %s\n", preparedText);

    for (int i = 0; i < strlen(preparedText); i += 2) {
        encryptPair(&preparedText[i], &preparedText[i + 1]);
    }

    printf("Encrypted Message: %s\n", preparedText);
}

int main() {
    char message[] = "Must see you over Cadogan West. Coming at once.";
    printf("Original Message: %s\n", message);

    encryptMessage(message);

    return 0;
}
```

Sample output: