# Cryptography and Network Security Lab
# Assignment 1

**1.1 Caesar Cipher**
**a) Implement Caesar Cipher Algorithm using C/C++/Java**

**CODE:**
**Encryption:**

```
#include<stdio.h>

#include<ctype.h>

int main() {

    char text[500], ch;

    int key;

    // Taking user input.
    printf("Enter a message to encrypt: ");

    scanf("%s", text);

    printf("Enter the key: ");

    scanf("%d", & key);

    // Visiting character by character.

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

        ch = text[i];
        // Check for valid characters.
        if (isalnum(ch)) {

            //Lowercase characters.
            if (islower(ch)) {
                ch = (ch - 'a' + key) % 26 + 'a';
            }
            // Uppercase characters.
            if (isupper(ch)) {
                ch = (ch - 'A' + key) % 26 + 'A';
            }

            // Numbers.
            if (isdigit(ch)) {
                ch = (ch - '0' + key) % 10 + '0';
            }
        }
        // Invalid character.
        else {
            printf("Invalid Message");
        }

        // Adding encoded answer.
        text[i] = ch;
```

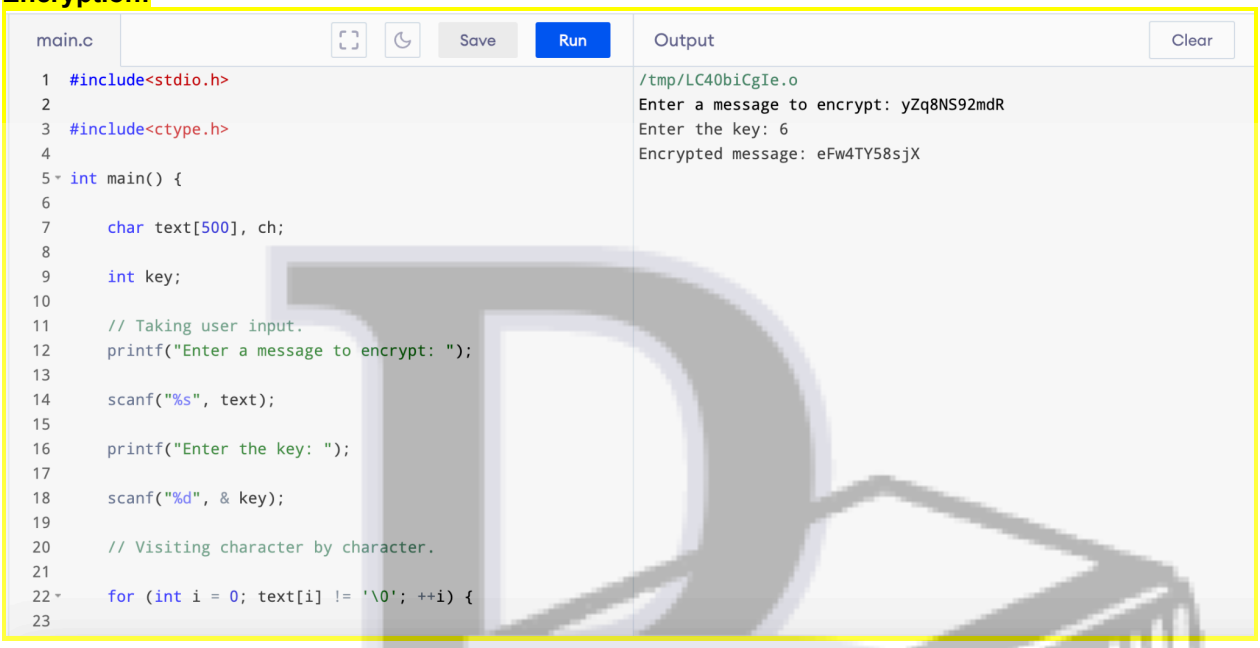
```
    }

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

    return 0;
}
```

**Decryption:**

```
#include<stdio.h>

#include<ctype.h>

int main() {

    char text[500], ch;

    int key;

    // Taking user input.

    printf("Enter a message to decrypt: ");

    scanf("%s", text);

    printf("Enter the key: ");

    scanf("%d", & key);

    // Visiting each character.
    for (int i = 0; text[i] != '\0'; ++i) {

        ch = text[i];
        // Check for valid characters.
        if (isalnum(ch)) {
            //Lowercase characters.
            if (islower(ch)) {
                ch = (ch - 'a' - key + 26) % 26 + 'a';
            }
            // Uppercase characters.
            if (isupper(ch)) {
                ch = (ch - 'A' - key + 26) % 26 + 'A';
            }
            // Numbers.
            if (isdigit(ch)) {
                ch = (ch - '0' - key + 10) % 10 + '0';
            }
        }
        // Invalid characters.
        else {
            printf("Invalid Message");
        }
        // Adding decoded character back.
        text[i] = ch;

    }

    printf("Decrypted message: %s", text);

    return 0;

}
```

## CODE SCREENSHOT:

## Encryption:



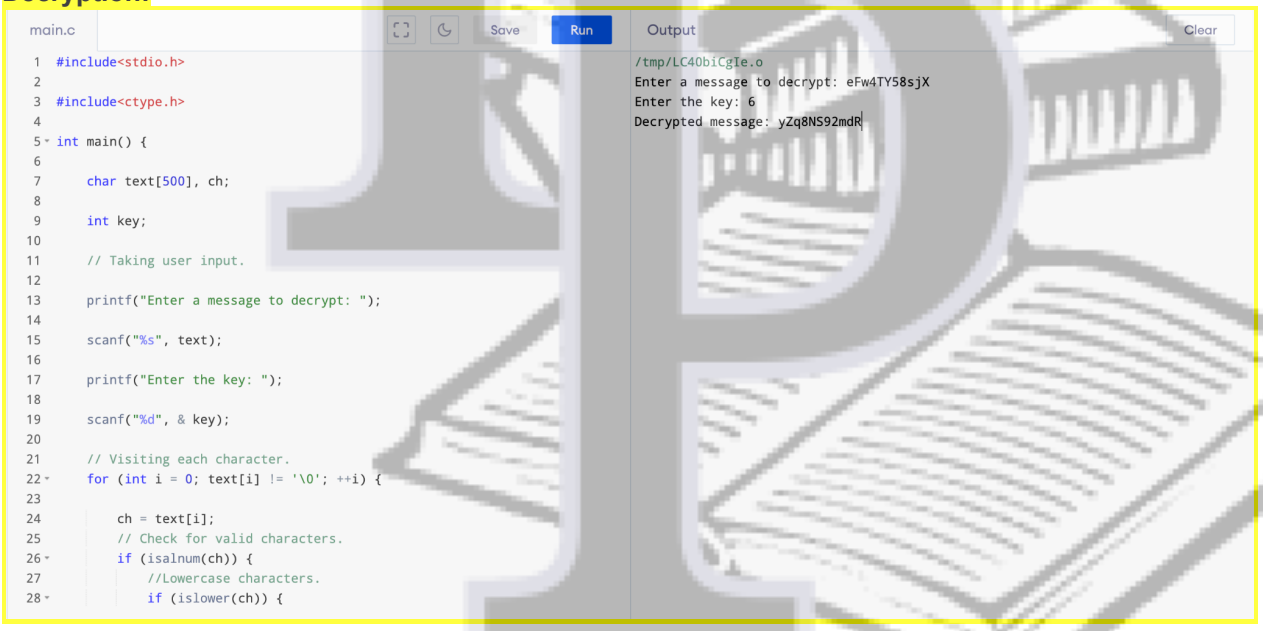
main.c

Save Run Output Clear

```c
#include<stdio.h>

#include<ctype.h>

int main() {

    char text[500], ch;

    int key;

    // Taking user input.
    printf("Enter a message to encrypt: ");

    scanf("%s", text);

    printf("Enter the key: ");

    scanf("%d", & key);

    // Visiting character by character.

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

```

```
/tmp/LC40biCgIe.o
Enter a message to encrypt: yZq8NS92mdR
Enter the key: 6
Encrypted message: eFw4TY58sjX
```

## Decryption:



main.c

Save Run Output Clear

```c
#include<stdio.h>

#include<ctype.h>

int main() {

    char text[500], ch;

    int key;

    // Taking user input.

    printf("Enter a message to decrypt: ");

    scanf("%s", text);

    printf("Enter the key: ");

    scanf("%d", & key);

    // Visiting each character.
    for (int i = 0; text[i] != '\0'; ++i) {

        ch = text[i];
        // Check for valid characters.
        if (isalnum(ch)) {
            //Lowercase characters.
            if (islower(ch)) {
```

```
/tmp/LC40biCgIe.o
Enter a message to decrypt: eFw4TY58sjX
Enter the key: 6
Decrypted message: yZq8NS92mdR
```

## OUTPUT SCREENSHOT:

## Encryption:

```
Output                                                    Clear

/tmp/LC40biCgIe.o
Enter a message to encrypt: yZq8NS92mdR
Enter the key: 6
Encrypted message: eFw4TY58sjX
```

**Decryption:**

```
Output                                                    Clear

/tmp/LC40biCgIe.o
Enter a message to decrypt: eFw4TY58sjX
Enter the key: 6
Decrypted message: yZq8NS92mdR
```

**b) Cipher Text is given below, Find the key and Plain text using Caesar Cipher.**

vnnc vn jocna cqn cxpj yjjach

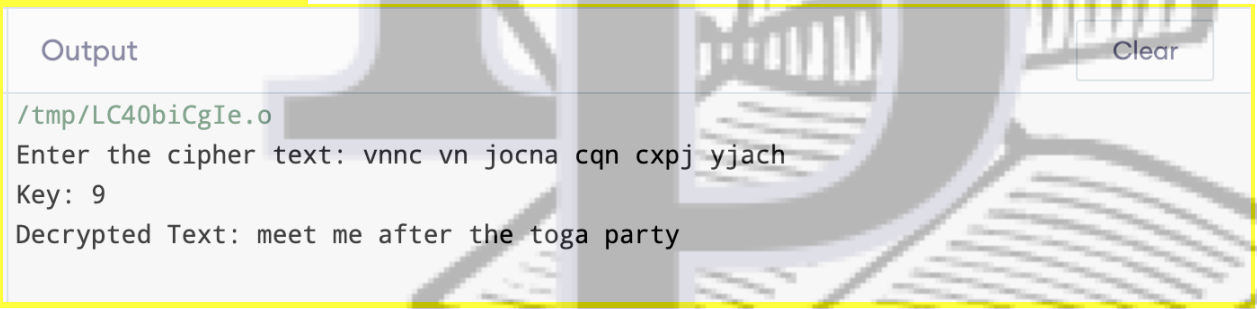
**CODE:**

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

// Function to perform Caesar Cipher decryption
void decrypt(char message[], int key) {
    for (int i = 0; message[i] != '\0'; ++i) {
        if (message[i] >= 'A' && message[i] <= 'Z') {
            message[i] = (message[i] - key - 'A' + 26) % 26 + 'A';
        } else if (message[i] >= 'a' && message[i] <= 'z') {
            message[i] = (message[i] - key - 'a' + 26) % 26 + 'a';
        }
    }
}

int main() {
    char cipherText[100];

    // Get input from the user
    printf("Enter the cipher text: ");
    fgets(cipherText, sizeof(cipherText), stdin);

    // Try all possible keys and print the correct decrypted message and key
    for (int key = 1; key <= 26; ++key) {
        char decryptedText[100];
        strcpy(decryptedText, cipherText);
        decrypt(decryptedText, key);

        // Check if the decrypted text looks like valid English
        if (strstr(decryptedText, "the") != NULL || strstr(decryptedText, "and") != NULL) {
            printf("Key: %d\nDecrypted Text: %s", key, decryptedText);
            break;  // Stop once a plausible decryption is found
```

```
    }
  }

  return 0;
}
```

**CODE SCREENSHOT:**


**OUTPUT SCREENSHOT:**




**1.2 Play Fair Cipher**
**Implement Play fair Cipher using c/C++/Java**

**CODE:**

**Encryption:**



```
// Implementation of Playfair Cipher in C program

#include <stdio.h>   //header file
#include <stdlib.h>  //header file
#include <string.h>  //header file

#define SIZE 30

// this function will convert the string to lowercase

void toLowerCase(char plain[], int ps)
{
```

```
    int i;
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}

// this function will remove all the spaces
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

// this function will generate the 5x5 grid square
void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    // character hashmap of 26 character that will
    // store count of the alphabet.
    dicty = (int*)calloc(26, sizeof(int));
    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }

    dicty['j' - 97] = 1;

    i = 0;
    j = 0;

    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }

    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
}

// this function will search for the characters of a digraph
// in the key and return position of key
void search(char keyT[5][5], char a, char b, int arr[])
```

```
{
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';

    for (i = 0; i < 5; i++) {

        for (j = 0; j < 5; j++) {

            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}

// this function will find the modulus with 5
int mod5(int a) { return (a % 5); }

// this function will make the plain text length even
int prepare(char str[], int ptrs)
{
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

// encryption will done using this function
void encrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] + 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] + 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

// this function will encrypt cipher text using Playfair Cipher algorithm
void encryptByPlayfairCipher(char str[], char key[])
```

```
{
    char ps, ks, keyT[5][5];


    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);


    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    encrypt(str, keyT, ps);
}

// main code
int main()
{
    char str[SIZE], key[SIZE];

    // key text
    strcpy(key, "Algorithm");
    printf("Key text: %s\n", key);

    // Plaintext
    strcpy(str, "Programming");
    printf("Plain text: %s\n", str);

    // encryption using the "Playfair Cipher" algorithmn
    encryptByPlayfairCipher(str, key);

    printf("Cipher text: %s\n", str);

    return 0;
}
```

**Decryption:**

```
#include <stdio.h>    // header file
#include <stdlib.h>   // header file
#include <string.h>   // header file
#define SIZE 30

// Conversion of characters
// of the string to lowercase
void toLowerCase(char plain[], int ps)
{
    int i;
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}


// spaces is removed from the string
int removeSpaces(char* plain, int ps)
```

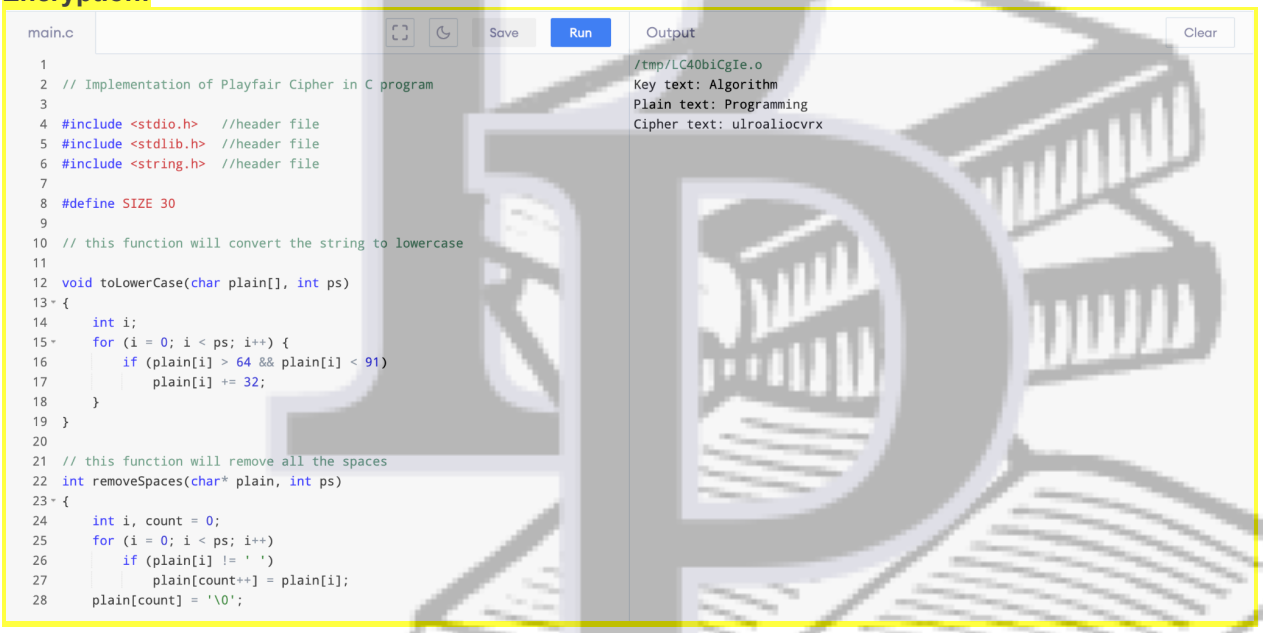
```
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

// here, 5x5 key square is generated
void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    // hashmap of 26 character that willl store the alphabet count
    dicty = (int*)calloc(26, sizeof(int));

    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }
    dicty['j' - 97] = 1;

    i = 0;
    j = 0;
    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
}

// characters and position of letters od digraph is searched from the key
void search(char keyT[5][5], char a, char b, int arr[])
{
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';

    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
```

```
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}

// this function will find the modulus with 5
int mod5(int a)
{
    if (a < 0)
        a += 5;
    return (a % 5);
}

// this function calls the decrypt function
void decrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];
    for (i = 0; i < ps; i += 2) {
        search(keyT, str[i], str[i + 1], a);
        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] - 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] - 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

// this function wil call the decrypt function
void decryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];

    // Key text
    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);

    // ciphertext
    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    generateKeyTable(key, ks, keyT);

    decrypt(str, keyT, ps);
}

// main code
int main()
{
    char str[SIZE], key[SIZE];
```

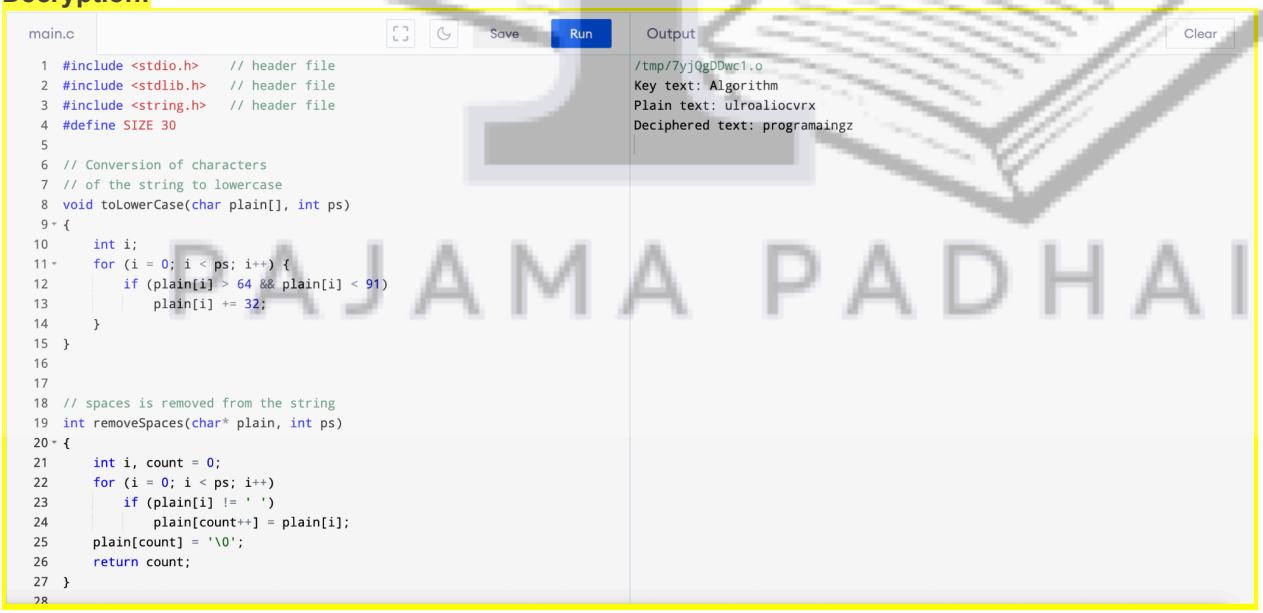
```
    // Key text that needs to be encrypted
    strcpy(key, "Algorithm");
    printf("Key text: %s\n", key);

    // Ciphertext that needs to be decrypted
    strcpy(str, "ulroaliocvrx");
    printf("Plain text: %s\n", str);

    // encryption is done using Playfair Cipher algorithm
    decryptByPlayfairCipher(str, key);

    printf("Deciphered text: %s\n", str);

    return 0;
}
```

## CODE SCREENSHOT:

### Encryption:




### Decryption:



**OUTPUT SCREENSHOT:**

**Encryption:**

```
Output                                                    Clear

/tmp/LC4ObiCgIe.o
Key text: Algorithm
Plain text: Programming
Cipher text: ulroaliocvrx
```

**Decryption:**

```
Output                                                    Clear

/tmp/7yjQgDDwc1.o
Key text: Algorithm
Plain text: ulroaliocvrx
Deciphered text: programaingz
```

**1.3 Vigenere Cipher**
**Implement Vigenere Cipher Algorithm using C/C++/Java**

**CODE:**

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

void encrypt() {
    char plaintext[128];
    char key[16];
    printf("\nEnter the plaintext (up to 128 characters): ");
    scanf(" %[^\n]", plaintext); // Read input with spaces
    printf("Enter the key (up to 16 characters): ");
    scanf(" %[^\n]", key);

    printf("Cipher Text: ");
    for (int i = 0, j = 0; i < strlen(plaintext); i++, j++) {
        if (j >= strlen(key)) {
            j = 0;
        }
        int shift = toupper(key[j]) - 'A';
        char encryptedChar = ((toupper(plaintext[i]) - 'A' + shift) % 26) + 'A';
        printf("%c", encryptedChar);
    }
    printf("\n");
}

void decrypt() {
    char ciphertext[128];
    char key[16];
    printf("\nEnter the ciphertext: ");
    scanf(" %[^\n]", ciphertext);
    printf("Enter the key: ");
    scanf(" %[^\n]", key);
```

```
    printf("Deciphered Text: ");
    for (int i = 0, j = 0; i < strlen(ciphertext); i++, j++) {
        if (j >= strlen(key)) {
            j = 0;
        }
        int shift = toupper(key[j]) - 'A';
        char decryptedChar = ((toupper(ciphertext[i]) - 'A' - shift + 26) % 26) + 'A';
        printf("%c", decryptedChar);
    }
    printf("\n");
}

int main() {
    int option;
    while (1) {
        printf("\n1. Encrypt");
        printf("\n2. Decrypt");
        printf("\n3. Exit\n");
        printf("\nEnter your option: ");
        scanf("%d", &option);

        switch (option) {
            case 1:
                encrypt();
                break;
            case 2:
                decrypt();
                break;
            case 3:
                exit(0);
            default:
                printf("\nInvalid selection! Try again.\n");
                break;
        }
    }
    return 0;
}
```

**CODE SCREENSHOT:**

```
main.c                                                    Save   Run

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

void encrypt() {
    char plaintext[128];
    char key[16];
    printf("\nEnter the plaintext (up to 128 characters): ");
    scanf(" %[^\n]", plaintext); // Read input with spaces
    printf("Enter the key (up to 16 characters): ");
    scanf(" %[^\n]", key);

    printf("Cipher Text: ");
    for (int i = 0, j = 0; i < strlen(plaintext); i++, j++) {
        if (j >= strlen(key)) {
            j = 0;
        }
        int shift = toupper(key[j]) - 'A';
        char encryptedChar = ((toupper(plaintext[i]) - 'A' + shift) % 26) +
            'A';
        printf("%c", encryptedChar);
    }
    printf("\n");
}

void decrypt() {
```

```
Output                                                    Clear

/tmp/7yjQgDDwc1.o
1. Encrypt
2. Decrypt
3. Exit

Enter your option: 1
Enter the plaintext (up to 128 characters): hello
Enter the key (up to 16 characters): 4
Cipher Text: ;8??B

1. Encrypt
2. Decrypt
3. Exit

Enter your option: 2
Enter the ciphertext: ;8??B
Enter the key: 4
Deciphered Text: HELLO

1. Encrypt
2. Decrypt
3. Exit

Enter your option:
```

**OUTPUT SCREENSHOT:**

Output Clear

```
/tmp/7yjQgDDwc1.o
1. Encrypt
2. Decrypt
3. Exit

Enter your option: 1
Enter the plaintext (up to 128 characters): hello
Enter the key (up to 16 characters): 4
Cipher Text: ;8??B

1. Encrypt
2. Decrypt
3. Exit

Enter your option: 2
Enter the ciphertext: ;8??B
Enter the key: 4
Deciphered Text: HELLO
```

**1.4 Hill Cipher**
**Implement Hill Cipher Algorithm using C/C++/Java**

**CODE:**

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

float encrypt[3][1], decrypt[3][1], a[3][3], b[3][3], mes[3][1], c[3][3];

void encryption(); //encrypts the message
void decryption(); //decrypts the message
void getKeyMessage(); //gets key and message from user
void inverse(); //finds inverse of key matrix

void main() {
getKeyMessage();
encryption();
decryption();
}

void encryption() {
int i, j, k;
for(i = 0; i < 3; i++)
for(j = 0; j < 1; j++)
for(k = 0; k < 3; k++)
encrypt[i][j] = encrypt[i][j] + a[i][k] * mes[k][j];
printf("\nEncrypted string is: ");
for(i = 0; i < 3; i++)
printf("%c", (char)(fmod(encrypt[i][0], 26) + 97));

}

void decryption() {
```

```
int i, j, k;
inverse();
for(i = 0; i < 3; i++)
for(j = 0; j < 1; j++)
for(k = 0; k < 3; k++)
decrypt[i][j] = decrypt[i][j] + b[i][k] * encrypt[k][j];
printf("\nDecrypted string is: ");
for(i = 0; i < 3; i++)
printf("%c", (char)(fmod(decrypt[i][0], 26) + 97));
printf("\n");
}

void getKeyMessage() {
int i, j;
char msg[3];

printf("Enter 3x3 matrix for key (It should be inversible):\n");
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++) {
scanf("%f", &a[i][j]);
c[i][j] = a[i][j];
}
printf("\nEnter a 3 letter string: ");
scanf("%s", msg);
for(i = 0; i < 3; i++)
mes[i][0] = msg[i] - 97;
}

void inverse() {
int i, j, k;
float p, q;
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++) {
if(i == j)
b[i][j]=1;
else
b[i][j]=0;
}
for(k = 0; k < 3; k++) {
for(i = 0; i < 3; i++) {
p = c[i][k];
q = c[k][k];
for(j = 0; j < 3; j++) {
if(i != k) {
c[i][j] = c[i][j]*q - p*c[k][j];
b[i][j] = b[i][j]*q - p*b[k][j];
}
}
}
}
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++)
b[i][j] = b[i][j] / c[i][i];
printf("\n\nInverse Matrix is:\n");
for(i = 0; i < 3; i++) {
for(j = 0; j < 3; j++)
printf("%d ", b[i][j]);
printf("\n");
}
}
```

**CODE SCREENSHOT:**

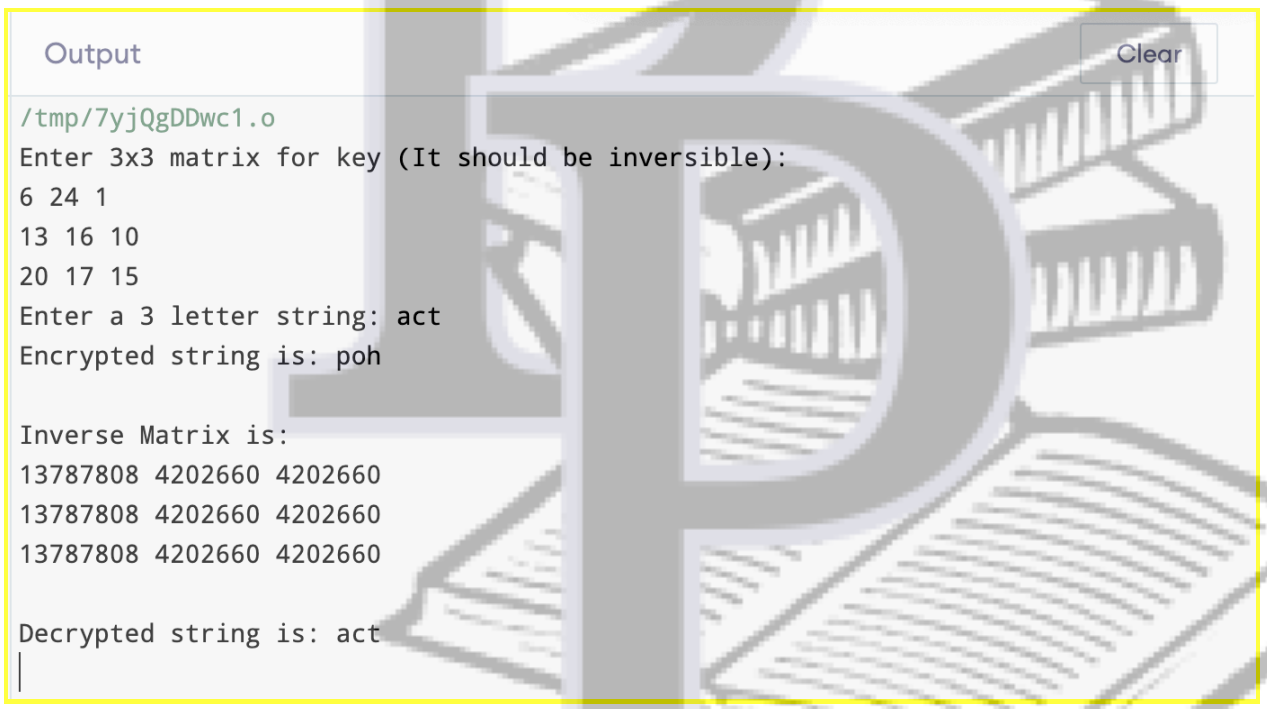
main.c

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

float encrypt[3][1], decrypt[3][1], a[3][3], b[3][3], mes[3][1], c[3][3];

void encryption(); //encrypts the message
void decryption(); //decrypts the message
void getKeyMessage(); //gets key and message from user
void inverse(); //finds inverse of key matrix

void main() {
getKeyMessage();
encryption();
decryption();
}

void encryption() {
int i, j, k;
for(i = 0; i < 3; i++)
for(j = 0; j < 1; j++)
for(k = 0; k < 3; k++)
encrypt[i][j] = encrypt[i][j] + a[i][k] * mes[k][j];
printf("\nEncrypted string is: ");
for(i = 0; i < 3; i++)
printf("%c", (char)(fmod(encrypt[i][0], 26) + 97));

}
```

Output

```
/tmp/7yjQgDDwc1.o
Enter 3x3 matrix for key (It should be inversible):
6 24 1
13 16 10
20 17 15
Enter a 3 letter string: act
Encrypted string is: poh

Inverse Matrix is:
13787808 4202660 4202660
13787808 4202660 4202660
13787808 4202660 4202660

Decrypted string is: act
```

**OUTPUT SCREENSHOT:**




**2. DES**

**Consider a sender and receiver who need to exchange data confidentially using symmetric encryption. Write program that implements DES encryption and decryption using a 64 bit key size and 64 bit block size**

**CODE:**

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

// Initial permutation table
const int initial_permutation[] = {
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1,
```

```
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
};

// Function to perform initial permutation
void initialPermutation(const uint8_t *input, uint8_t *output) {
    for (int i = 0; i < 64; ++i) {
        int bit = (initial_permutation[i] - 1) % 8;
        int byte = (initial_permutation[i] - 1) / 8;
        output[i / 8] |= ((input[byte] >> bit) & 1) << (7 - i % 8);
    }
}

// Function to perform DES encryption
void desEncrypt(const uint8_t *plainText, const uint8_t *key, uint8_t *cipherText) {
    // Initial permutation
    uint8_t permutedText[8] = {0};
    initialPermutation(plainText, permutedText);

    // Key generation and initial permutation
    uint8_t permutedKey[8] = {0};
    initialPermutation(key, permutedKey);

    // XOR with the key
    for (int i = 0; i < 8; ++i) {
        permutedText[i] ^= permutedKey[i];
    }

    // Display the encrypted text
    printf("Encrypted Text: ");
    for (int i = 0; i < 8; ++i) {
        printf("%02X ", permutedText[i]);
    }
    printf("\n");
}

int main() {
    uint8_t key[8];
    uint8_t plainText[8];

    // Get the key from the user
    printf("Enter the 64-bit key (8 characters): ");
    for (int i = 0; i < 8; ++i) {
        scanf("%2hhx", &key[i]);
    }

    // Get the plaintext from the user
    printf("Enter the 64-bit plaintext (8 characters): ");
    for (int i = 0; i < 8; ++i) {
        scanf("%2hhx", &plainText[i]);
    }

    // Perform DES encryption
    desEncrypt(plainText, key, NULL);

    return 0;
}
```

**CODE SCREENSHOT:**

main.c

Save Run

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

// Initial permutation table
const int initial_permutation[] = {
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
};

// Function to perform initial permutation
void initialPermutation(const uint8_t *input, uint8_t *output) {
    for (int i = 0; i < 64; ++i) {
        int bit = (initial_permutation[i] - 1) % 8;
        int byte = (initial_permutation[i] - 1) / 8;
        output[i / 8] |= ((input[byte] >> bit) & 1) << (7 - i % 8);
    }
}

// Function to perform DES encryption
void desEncrypt(const uint8_t *plainText, const uint8_t *key, uint8_t
    *cipherText) {
    // Initial permutation
```

Output Clear

```
/tmp/7yjQgDDwc1.o
Enter the 64-bit key (8 characters): 133457799BBCDFF1
Enter the 64-bit plaintext (8 characters): 0123456789ABCDEF
Encrypted Text: FF 88 00 00 22 AA FF 00
```

**OUTPUT SCREENSHOT:**