Institute of Science and Technology
Tribhuvan University

Department of computer science and Information Technology
Prithivi Narayan Campus, Pokhara

Lab report on

Cryptography (CSC327)

Submitted To :
Mr. Dev Timilsina

Submitted By :
Sabin Paudel

Symbol No:
79011904

Lab Index

~~~~~~

Name: Sabin Paudel Roll. No: 56

Level: Bachelors Year/Sem: 3rd/5th

Subject: Cryptography Faculty: B.Sc. CSIT
Course No: CSC 327

Lab.no	Experiment name	Submitted Date	Remarks
1.	To study the Caesar Cipher Technique..	2082-01-03	
2.	Compute the GCD of two numbers using the Euclidean algorithm.	2082-01-03	
3.	Implementation of the Diffie-Hellman key exchange algorithm in C.	2082-01-03	
4.	Determine if the two numbers are co-prime OR not.	2082-01-03	
5.	To study the Hill Cipher Encryption and Decryption and Vigenere Cipher.	2082-01-03	
6.	To study Monoalphabetic and Rail Fence Cipher technique.	2082-01-03	
~~~~~~

EXPERIMENT 1
TO STUDY THE CAESAR CIPHER..

1. OBJECTIVE

- To study the Caesar Cipher.

2. THEORY

The **Caesar Cipher** is one of the simplest and oldest encryption techniques. It is a type of **substitution cipher** in which each letter in the plaintext is **shifted a fixed number of positions** down the alphabet. The number of positions to shift is called the **key**.
For example, with a key of 3:

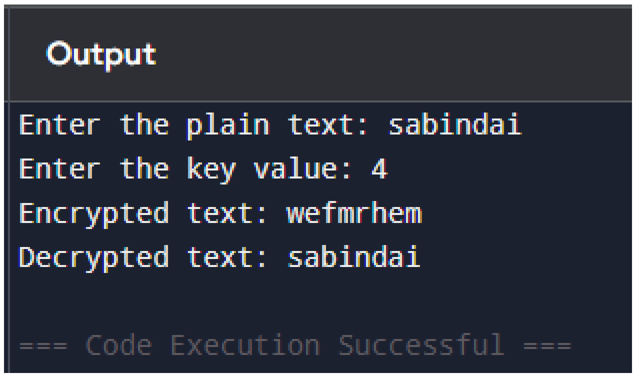
- A → D
- B → E
- C → F

... and so on
After 'Z', the cipher wraps around to 'A'. The same logic applies to lowercase letters.
To **decrypt** the message, the letters are shifted **backward** by the same key value.
Caesar Cipher only works on alphabetic characters and is **vulnerable to brute force attacks**, as there are only 25 possible keys

3. DEMONSTRATION:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main() {
    char plain_text[100], cipher_text[100];
    int key, i, length;
    printf("Enter the plain text: ");
    scanf("%s", plain_text);
    printf("Enter the key value: ");
    scanf("%d", &key);
    length = strlen(plain_text)
 // Encryption
    printf("Encrypted text: ");
    for (i = 0; i < length; i++) {
        char ch = plain_text[i];
        if (isupper(ch)) {
            cipher_text[i] = ((ch - 'A' + key) % 26) + 'A';
        } else if (islower(ch)) {
            cipher_text[i] = ((ch - 'a' + key) % 26) + 'a';
        } else {
            cipher_text[i] = ch; // Non-alphabetic characters unchanged  }
        printf("%c", cipher_text[i]);   }
    cipher_text[length] = '\0';
    // Decryption
    printf("\nDecrypted text: ");
    for (i = 0; i < length; i++) {
        char ch = cipher_text[i];
        if (isupper(ch)) {
            plain_text[i] = ((ch - 'A' - key + 26) % 26) + 'A';
        } else if (islower(ch)) {
            plain_text[i] = ((ch - 'a' - key + 26) % 26) + 'a';
        } else {
            plain_text[i] = ch;  }
        printf("%c", plain_text[i]); }
    plain_text[length] = '\0';
    return 0;
}
```

4. OUTPUT



Output

```
Enter the plain text: sabindai
Enter the key value: 4
Encrypted text: wefmrhem
Decrypted text: sabindai


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

5. CONCLUSION

The Caesar Cipher demonstrates the basic principles of encryption using simple character substitution. Although it is not secure for modern use due to its limited key space, it serves as a foundational concept in cryptography. Through this lab, we understood how encryption and decryption work using character shifting, and how important key management is in secure communication.

EXPERIMENT 2
COMPUTE THE GCD OF TWO NUMBERS USING THE EUCLIDEAN ALGORITHM.

1. OBJECTIVE:

- Compute the GCD of two numbers using the Euclidean algorithm.

2. THEORY:

The Greatest Common Divisor (GCD) of two numbers is the largest number that divides both of them without leaving a remainder. The Euclidean Algorithm is an efficient method to compute the GCD based on the principle that:

GCD(a, b) = GCD(b, a % b)

where a > b, and % is the modulo operator.

The process continues until the remainder becomes 0. At that point, the non-zero divisor is the GCD. Example:

To find GCD of 48 and 18:GCD(48, 18)

= GCD(18, 48 % 18) = GCD(18, 12)

= GCD(12, 18 % 12) = GCD(12, 6)

= GCD(6, 12 % 6) = GCD(6, 0)

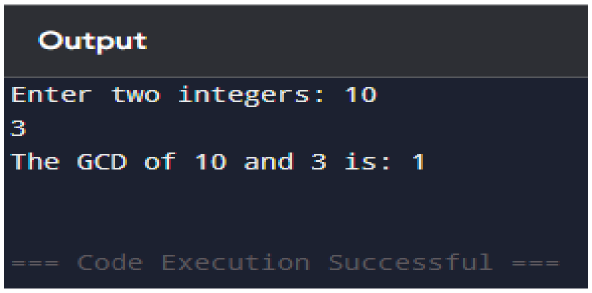
→ GCD is 6

This method is fast, uses repeated division, and works well for both small and large integers.

3. DEMONSTRATION:

```
// Function to compute the GCD of two numbers using the Euclidean algorithm
#include <stdio.h>
int gcd(int a, int b) {
while (b != 0) {
int temp = b;b = a % b;a = temp;}
return a;}
int main() {
int num1, num2;
printf("Enter two integers: ");
scanf("%d %d", &num1, &num2);
int result = gcd(num1, num2);
printf("The GCD of %d and %d is: %d\n", num1, num2, result);
return 0;
}
```

4. OUTPUT:



```
Output

Enter two integers: 10
3
The GCD of 10 and 3 is: 1


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

5. CONCLUSION:

The Euclidean Algorithm provides a simple and efficient method to calculate the GCD of two numbers using repeated division. It significantly reduces the number of operations compared to other methods. Through this lab, we understood how the algorithm uses the modulo operation to find the GCD and why it is preferred in mathematical computations and programming.

EXPERIMENT 3
IMPLEMENTATION OF THE DIFFIE-HELLMAN KEY EXCHANGE ALGORITHM IN C.

1. OBJECTIVES

- Implementation of the Diffie-Hellman key exchange algorithm in C.

2. THEORY:

The **Diffie-Hellman key exchange algorithm** is a method used to securely exchange cryptographic keys over a public channel. It was developed by **Whitfield Diffie** and **Martin Hellman** in 1976 and is considered one of the first practical implementations of **public-key cryptography**.

The core idea is that two parties can generate a **shared secret key** using the following steps:

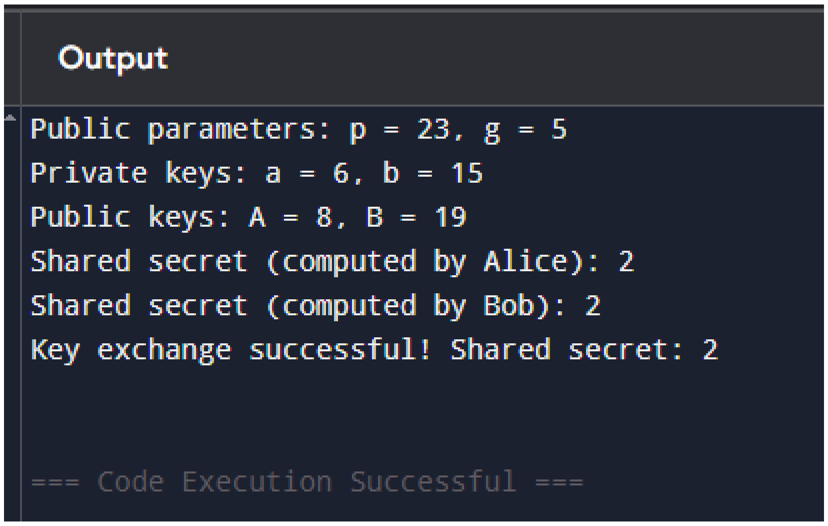
- Both parties agree on a **large prime number (P)** and a **primitive root (G)**.
- Each party selects a **private key (a and b)**, which is kept secret.
- They compute their **public keys** as:
  A = G^a mod P
  B = G^b mod P
- They exchange public keys over the network.
- Each party then computes the **shared secret key** using the other's public key:
- Shared Key = (B^a) mod P = (A^b) mod P

Even though the public values (P, G, A, B) are exchanged openly, it is computationally difficult to determine the shared secret key without knowing the private keys. This method is widely used in secure communications protocols such as HTTPS, SSH, and VPNs.

3. DEMONSTRATION:

```
#include <stdio.h>
#include <math.h>
long long int mod_exp(long long int base, long long int exponent, long long int modulus) {
long long int result = 1; // Function to perform modular exponentiation
base = base % modulus; // It returns (base^exponent) % modulus
while (exponent > 0) {
// If exponent is odd, multiply base with result
if (exponent % 2 == 1) {
result = (result * base) % modulus;} // exponent must be even now
exponent = exponent >> 1; // Divide the exponent by 2
base = (base * base) % modulus; // Square the base}
return result;}
int main() {
long long int p, g; // Public parameters (prime number and base)
long long int a, b; // Private keys
long long int A, B; // Public keys
long long int shared_secret_A, shared_secret_B; // Shared secret keys
// Example values for p and g
p = 23; // A prime number
g = 5; // A primitive root modulo p
// Private keys (these would normally be chosen randomly and kept secret)
a = 6; // Private key for Alice
b = 15; // Private key for Bob
// Compute the public keys
A = mod_exp(g, a, p); // A = g^a % p
B = mod_exp(g, b, p); // B = g^b % p
shared_secret_A = mod_exp(B, a, p); // s_A = B^a % p
shared_secret_B = mod_exp(A, b, p); // s_B = A^b % p
// Display results
printf("Public parameters: p = %lld, g = %lld\n", p, g);
printf("Private keys: a = %lld, b = %lld\n", a, b);
printf("Public keys: A = %lld, B = %lld\n", A, B);
printf("Shared secret (computed by Alice): %lld\n", shared_secret_A);
printf("Shared secret (computed by Bob): %lld\n", shared_secret_B);
// Both shared secrets should be the same
if (shared_secret_A == shared_secret_B) {
printf("Key exchange successful! Shared secret: %lld\n", shared_secret_A);
} else {
printf("Key exchange failed!\n");}
return 0;}
```

4. **OUTPUT:**



Output

```
Public parameters: p = 23, g = 5
Private keys: a = 6, b = 15
Public keys: A = 8, B = 19
Shared secret (computed by Alice): 2
Shared secret (computed by Bob): 2
Key exchange successful! Shared secret: 2


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

5. **CONCLUSION:**

The implementation of the Diffie-Hellman Key Exchange in C demonstrates how two parties can securely generate a common secret key over an insecure channel. This lab helped understand the mathematical operations involved in secure key exchange and the importance of using prime numbers and modular arithmetic. The successful key exchange confirms the correctness of the implementation and its potential use in real-world secure communication systems.

EXPERIMENT 4
DETERMINE IF THE TWO NUMBERS ARE CO-PRIME OR NOT.

1. **OBJECTIVES:**
   - Determine if the two numbers are co-prime OR not.

2. **THEORY:**

Two integers are said to be co-prime (or relatively prime) if their Greatest Common Divisor (GCD) is 1. This means they have no common factors other than 1.

To check if two numbers are co-prime:

- Compute their GCD using the Euclidean algorithm, which repeatedly replaces the larger number with the remainder until the remainder is 0.
- If the final GCD is 1, the numbers are co-prime.

Example:

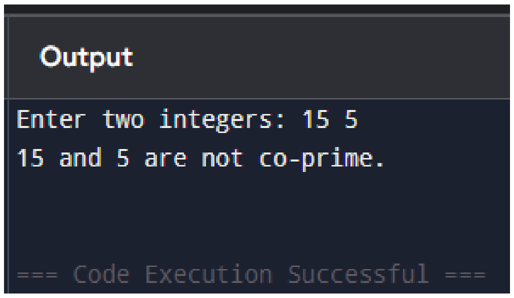
- 8 and 15 → GCD(8, 15) = 1 → Co-prime
- 12 and 18 → GCD(12, 18) = 6 → Not co-prime

This method is efficient and widely used in number theory, especially in cryptographic algorithms like RSA.

3. **DEMONSTRATION:**

```
#include <stdio.h>
int gcd(int a, int b) {
while (b != 0) {
int temp = b;b = a % b;a = temp;}
return a;} // Function to check if two numbers are co-prime
int are_coprime(int num1, int num2) {
return gcd(num1, num2) == 1;}
int main() {
int num1, num2;
// Input two numbers
printf("Enter two integers: ");
scanf("%d %d", &num1, &num2);
if (are_coprime(num1, num2)) // Check if the numbers are co-prime {
printf("%d and %d are co-prime.\n", num1, num2);} else {
printf("%d and %d are not co-prime.\n", num1, num2);}
return 0;}
```

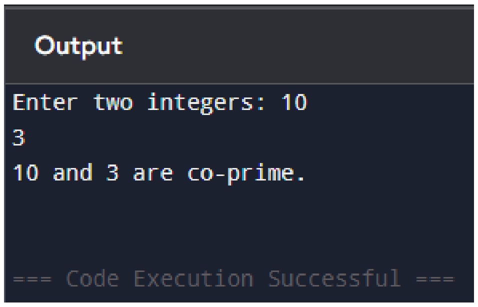
4. **OUTPUT:**



Output

```
Enter two integers: 15 5
15 and 5 are not co-prime.


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



Output

```
Enter two integers: 10
3
10 and 3 are co-prime.


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

5. **CONCLUSION:**

The program successfully checks if two given numbers are co-prime using the Euclidean algorithm. This lab enhanced our understanding of number theory concepts such as GCD and co-primality, which are essential in many cryptographic applications. The Euclidean algorithm offers a simple and effective approach to solving this problem.

EXPERIMENT 5
TO STUDY THE HILL CIPHER ENCRYPTION AND DECRYPTION AND VIGENERE CIPHER.

1. OBJECTIVE:

- To study the Hill Cipher Encryption and Decryption and Vigenere Cipher.

2. THEORY:

- **Hill Cipher**

The **Hill Cipher** is a **polyalphabetic substitution cipher** based on **linear algebra**. It was invented by **Lester S. Hill** in 1929. The core concept is to use **matrix multiplication** to encrypt blocks of plaintext letters.

- A **key matrix (K)** of size n x n is chosen, where n is the block size.
- The plaintext is divided into blocks of size n, converted to numeric values (A=0, B=1, ..., Z=25), and multiplied with the key matrix:
  $C = (K \times P) \bmod 26$
- Decryption is performed using the **inverse of the key matrix**:
  $P = (K^{-1} \times C) \bmod 26$
- The key matrix must be **invertible mod 26**, i.e., it should have a non-zero determinant that is relatively prime to 26.

Hill cipher is secure for small blocks but is vulnerable to known plaintext attacks if enough ciphertext is available.

- **Vigenère Cipher**

The Vigenère Cipher is a repeating key polyalphabetic cipher. It encrypts alphabetic text by shifting each letter based on a repeating keyword.

- Encryption formula:
  C[i] = (P[i] + K[i % key_length]) mod 26

- Decryption formula:
  P[i] = (C[i] - K[i % key_length] + 26) mod 26
- Here, P is the plaintext, C is the ciphertext, and K is the key converted to numbers (A=0 to Z=25).

The Vigenère Cipher was considered very secure for centuries and is stronger than Caesar Cipher due to the use of multiple Caesar shifts. However, it can be broken using frequency analysis and Kasiski examination if the key is short or reused.

3. Demonstration:

a. Hill Cipher Encryption and Decryption

```
#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 getKeyMessage();    // Gets key and message from user
void encryption();       // Encrypts the message
void decryption();       // Decrypts the message
void inverse();          // Finds inverse of the key matrix
int main() {
    getKeyMessage();
    encryption();
    decryption();
    return 0;
}
void getKeyMessage() {
    int i, j;
    char msg[3];
    printf("Enter 3x3 matrix for key (It should be invertible):\n");
    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++) {
            scanf("%f", &a[i][j]);
            c[i][j] = a[i][j]; // Copy for inverse
        }
    printf("Enter a 3-letter string (lowercase): ");
    scanf("%s", msg);
    for(i = 0; i < 3; i++)
        mes[i][0] = msg[i] - 97;
}
void encryption() {
    int i, j, k;
    for(i = 0; i < 3; i++)
        for(j = 0; j < 1; j++) {
            encrypt[i][j] = 0;
            for(k = 0; k < 3; k++) {
                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(); // Calculate inverse of the matrix
    for(i = 0; i < 3; i++)
        for(j = 0; j < 1; j++) {
            decrypt[i][j] = 0;
            for(k = 0; k < 3; k++) {
                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 inverse() {
    int i, j, k;
    float p, q;
    // Initialize b as identity matrix
    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
            b[i][j] = (i == j) ? 1 : 0;
```

```
// Perform Gauss-Jordan elimination
for(k = 0; k < 3; k++) {
    for(i = 0; i < 3; i++) {
        if(i != k) {
            p = c[i][k];
            q = c[k][k];
            for(j = 0; j < 3; j++) {
                c[i][j] = c[i][j] * q - c[k][j] * p;
                b[i][j] = b[i][j] * q - b[k][j] * p;
            }}}  }
// Normalize the diagonal to 1
for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
        b[i][j] = b[i][j] / c[i][i];
printf("\nInverse Matrix is:\n");
for(i = 0; i < 3; i++) {
    for(j = 0; j < 3; j++)
        printf("%0.2f ", b[i][j]);
    printf("\n");
}}
```

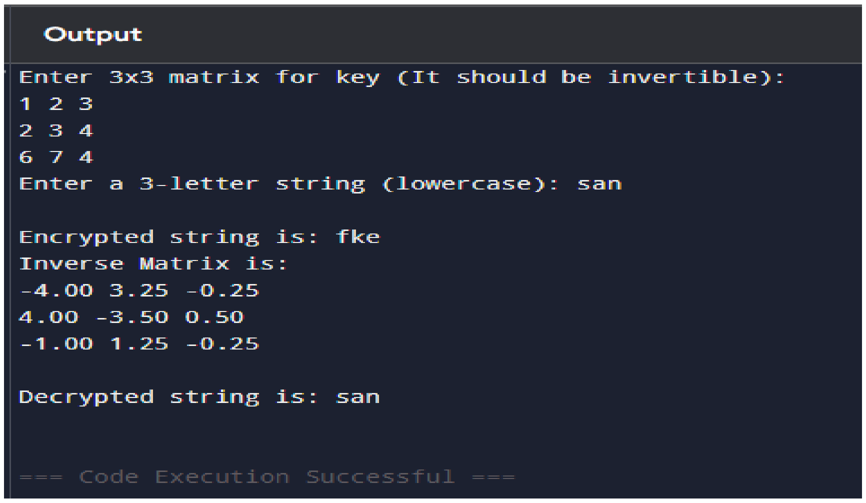
b. Vigenère Cipher Encryption and Decryption.

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>  // for exit()
#include <string.h>
// Function prototypes
void encipher();
void decipher();
int main() {
    int option;
    while (1) {
        printf("\n======= Vigenère Cipher =======");
        printf("\n1. Encipher");
        printf("\n2. Decipher");
        printf("\n3. Exit");
        printf("\nEnter your option: ");
        scanf("%d", &option);
        getchar(); // clear newline character from input buffer
        switch (option) {
            case 1:
                encipher();
                break;
            case 2:
                decipher();
                break;
            case 3:
                exit(0);
            default:
```

```
            printf("\nInvalid selection! Try again.");} }
    return 0;
}
void encipher() {
    unsigned int i, j;
    char plain[128], key[16];
    printf("\nEnter the plaintext (max 127 characters): ");
    fgets(plain, sizeof(plain), stdin);
    plain[strcspn(plain, "\n")] = '\0'; // remove newline
    printf("Enter the key (max 15 characters): ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = '\0'; // remove newline
    printf("Encrypted text: ");
    for (i = 0, j = 0; i < strlen(plain); i++) {
        if (!isalpha(plain[i])) {
            printf("%c", plain[i]);  // skip encryption for non-alphabet
            continue;  }
        if (j >= strlen(key)) j = 0;
        // Encrypt using Vigenère formula: Ci = (Pi + Ki) % 26
        printf("%c",
            (char)(65 + (((toupper(plain[i]) - 65) + (toupper(key[j]) - 65)) % 26))  );
        j++;   }
    printf("\n");}
void decipher() {
    unsigned int i, j;
    char cipher[128], key[16];
    int value;
    printf("\nEnter the ciphertext: ");
    fgets(cipher, sizeof(cipher), stdin);
    cipher[strcspn(cipher, "\n")] = '\0'; // remove newline
    printf("Enter the key: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = '\0'; // remove newline
    printf("Decrypted text: ");
    for (i = 0, j = 0; i < strlen(cipher); i++) {
        if (!isalpha(cipher[i])) {
            printf("%c", cipher[i]);
            continue;     }
        if (j >= strlen(key)) j = 0;
        // Decrypt using: Pi = (Ci - Ki + 26) % 26
        value = (toupper(cipher[i]) - 65) - (toupper(key[j]) - 65);
        if (value < 0) value += 26;
        printf("%c", (char)(65 + (value % 26)));
        j++; }
    printf("\n");}
```

4. OUTPUT:

a. Hill Cipher



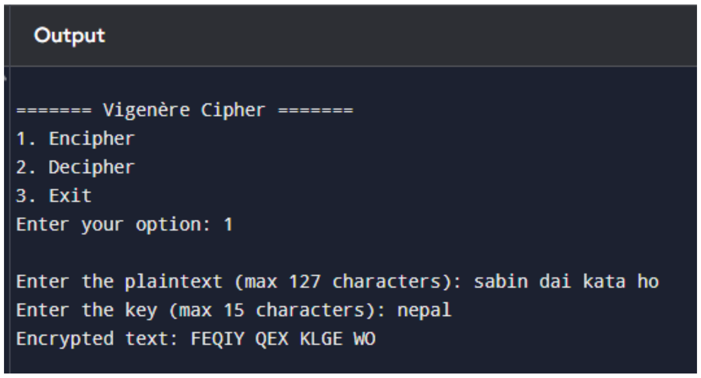
```
Output
Enter 3x3 matrix for key (It should be invertible):
1 2 3
2 3 4
6 7 4
Enter a 3-letter string (lowercase): san

Encrypted string is: fke
Inverse Matrix is:
-4.00 3.25 -0.25
4.00 -3.50 0.50
-1.00 1.25 -0.25

Decrypted string is: san


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

b. Vigenère Cipher

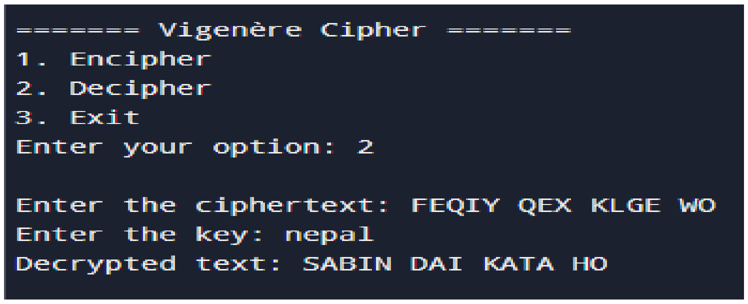


```
Output

======= Vigenère Cipher =======
1. Encipher
2. Decipher
3. Exit
Enter your option: 1

Enter the plaintext (max 127 characters): sabin dai kata ho
Enter the key (max 15 characters): nepal
Encrypted text: FEQIY QEX KLGE WO
```

Encryption



```
======= Vigenère Cipher =======
1. Encipher
2. Decipher
3. Exit
Enter your option: 2

Enter the ciphertext: FEQIY QEX KLGE WO
Enter the key: nepal
Decrypted text: SABIN DAI KATA HO
```

Decryption

5. CONCLUSION:

From this lab, we demonstrated the working principles of both the Hill Cipher and Vigenère Cipher. The Hill Cipher utilized matrix multiplication and modular arithmetic for encrypting and decrypting messages using a 3x3 key matrix, while the Vigenère Cipher implemented polyalphabetic substitution with a repeating key. Through this, we understood how classical encryption techniques function and how keys play a vital role in securing communication. This lab helped us build foundational knowledge of cryptography and its historical importance.

EXPERIMENT 6
TO STUDY MONOALPHABETIC AND RAIL FENCE CIPHER TECHNIQUE.

1. OBJECTIVE

- To study Monoalphabetic and Rail Fence Cipher technique.

2. THEORY

- **Monoalphabetic Cipher**

The **Monoalphabetic Cipher** is a type of substitution cipher in which each letter in the plaintext is replaced with another letter from a fixed substitution alphabet. The simplest form is the Caesar cipher, where each letter is shifted by a certain number. However, in a more general Monoalphabetic Cipher, the mapping between letters of the alphabet and their substitutes is arbitrary and fixed. This type of cipher relies on a one-to-one correspondence between the letters in the plaintext and ciphertext. The security of the Monoalphabetic Cipher is relatively weak because frequency analysis can reveal patterns in the ciphertext, which allows attackers to break the encryption by identifying common letters and letter combinations.

- **Rail Fence Cipher**

The Rail Fence Cipher is a form of transposition cipher where the letters of the plaintext are arranged in a zigzag pattern across multiple "rails" (or rows). The message is written in a zigzag manner on a number of rails and then read off row by row to form the ciphertext. The number of rails is a key factor in the encryption process, and the security increases with the number of rails. The Rail Fence Cipher is a relatively simple technique to implement, but it can be easily cracked through various decryption methods if the number of rails is known or guessed. Despite its simplicity, the Rail Fence Cipher demonstrates the concept of transposition, where the positions of the characters in the plaintext are altered, instead of substituting one character for another.

3. DEMONSTRATION

- **Monoalphabetic Cipher**

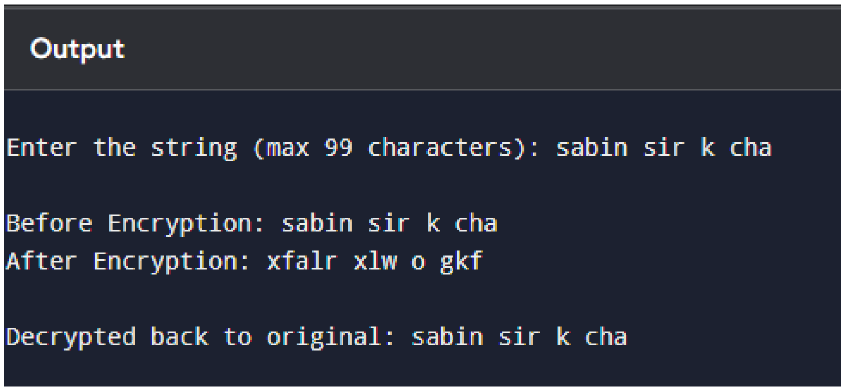
```
#include<stdio.h>
#include<string.h>
char monocipher_encr(char);
char monocipher_decr(char);
char alpha[27][3] = {
    { 'a', 'f' }, { 'b', 'a' }, { 'c', 'g' }, { 'd', 'u' }, { 'e', 'n' },
    { 'f', 'i' }, { 'g', 'j' }, { 'h', 'k' }, { 'i', 'l' }, { 'j', 'm' },
    { 'k', 'o' }, { 'l', 'p' }, { 'm', 'q' }, { 'n', 'r' }, { 'o', 's' },
    { 'p', 't' }, { 'q', 'v' }, { 'r', 'w' }, { 's', 'x' }, { 't', 'y' },
    { 'u', 'z' }, { 'v', 'b' }, { 'w', 'c' }, { 'x', 'd' }, { 'y', 'e' },
    { 'z', 'h' }
};
int main() {
    char str[100], str2[100];
    int i;
    printf("\nEnter the string (max 99 characters): ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = 0;  // Remove the newline character from fgets input
    // Encrypt the message
    for (i = 0; str[i]; i++) {
        str2[i] = monocipher_encr(str[i]);
    }
    str2[i] = '\0';
    printf("\nBefore Encryption: %s", str);
    printf("\nAfter Encryption: %s\n", str2);
    // Decrypt the message
    char str3[100];
    for (i = 0; str2[i]; i++) {
        str3[i] = monocipher_decr(str2[i]); }
    str3[i] = '\0';
    printf("\nDecrypted back to original: %s\n", str3);
    return 0;}
char monocipher_encr(char a) {
    int i;
    for (i = 0; i < 27; i++) {
        if (a == alpha[i][0]) {
            return alpha[i][1];   }}
    return a;  // Return character unchanged if it's not in the alphabet (e.g., space or
punctuation)
char monocipher_decr(char a) {
    int i;
    for (i = 0; i < 27; i++) {
        if (a == alpha[i][1]) {
            return alpha[i][0];   }   }
    return a;  // Return character unchanged if it's not in the alphabet
}
```

- **Rail Fence Cipher**

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void encryptRailFence(char *str, int rails);
void decryptRailFence(char *str, int rails);
int main() {
    int option, rails;
    char str[1000];
    while (1) {
        printf("\n1. Encrypt using Rail Fence Cipher");
        printf("\n2. Decrypt using Rail Fence Cipher");
        printf("\n3. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &option);
        // Clear the input buffer
        getchar();
        if (option == 1) {
            printf("\nEnter a Secret Message: ");
            fgets(str, sizeof(str), stdin);
            str[strcspn(str, "\n")] = 0;  // Remove trailing newline
            printf("Enter number of rails: ");
            scanf("%d", &rails);
            encryptRailFence(str, rails);
        }
        else if (option == 2) {
            printf("\nEnter the Ciphertext: ");
            fgets(str, sizeof(str), stdin);
            str[strcspn(str, "\n")] = 0;  // Remove trailing newline
            printf("Enter number of rails: ");
            scanf("%d", &rails);
            decryptRailFence(str, rails); }
        else if (option == 3) {
            break;  }
        else {
            printf("\nInvalid choice! Try again.");  } }
    return 0;}
void encryptRailFence(char *str, int rails) {
    int len = strlen(str);
    char rail[rails][len];
    int i, j, row, col;
    // Initialize the rail matrix
    for (i = 0; i < rails; i++) {
        for (j = 0; j < len; j++) {
            rail[i][j] = '\n';  // Mark the empty spaces
        }    }
    // Fill the rail matrix with the message
    row = 0;  col = 0;
    bool down = false;
    for (i = 0; i < len; i++) {
        rail[row][col++] = str[i];
        if (row == 0 || row == rails - 1) {
            down = !down; }
        row = down ? row + 1 : row - 1;  }
    // Print the encrypted message
    printf("\nEncrypted Message: ");
    for (i = 0; i < rails; i++) {
        for (j = 0; j < len; j++) {
            if (rail[i][j] != '\n') {
                printf("%c", rail[i][j]);    }    }  }
    printf("\n");}
void decryptRailFence(char *str, int rails) {
    int len = strlen(str);
    char rail[rails][len];
    int i, j, row, col;
    // Initialize the rail matrix
    for (i = 0; i < rails; i++) {
        for (j = 0; j < len; j++) {
            rail[i][j] = '\n';  // Mark the empty spaces
    }  }
    // Create a pattern to place characters in the rail matrix
    row = 0; col = 0;
    bool down = false;
    for (i = 0; i < len; i++) {
        rail[row][col++] = '*';  // Mark the path where characters should be placed
        if (row == 0 || row == rails - 1) {
            down = !down;  }
        row = down ? row + 1 : row - 1; }
    // Place the characters from the ciphertext into the rail matrix
    col = 0;
    for (i = 0; i < rails; i++) {
        for (j = 0; j < len; j++) {
            if (rail[i][j] == '*' && col < len) {
                rail[i][j] = str[col++];   }   }   }
    // Read the plaintext from the rail matrix
    printf("\nDecrypted Message: ");
    row = 0;   col = 0;  down = false;
    for (i = 0; i < len; i++) {
        printf("%c", rail[row][col++]);
        if (row == 0 || row == rails - 1) {
            down = !down; }
        row = down ? row + 1 : row - 1;  }
    printf("\n");}
```

4. **OUTPUT**

   Monoalphabetic Cipher



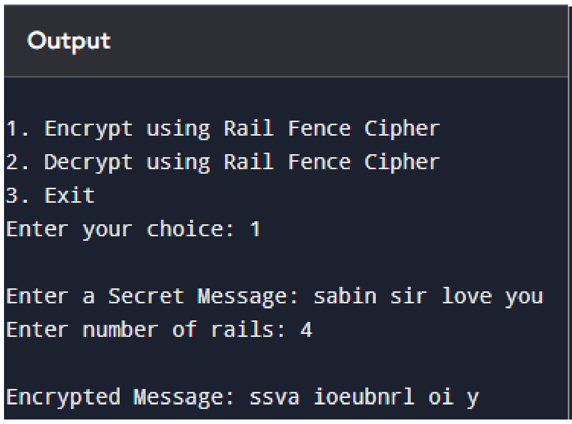
```
Output

Enter the string (max 99 characters): sabin sir k cha

Before Encryption: sabin sir k cha
After Encryption: xfalr xlw o gkf

Decrypted back to original: sabin sir k cha
```

Rail Fence Cipher

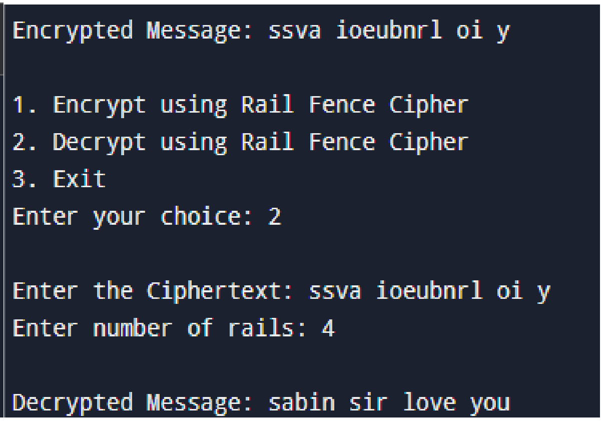


```
Output

1. Encrypt using Rail Fence Cipher
2. Decrypt using Rail Fence Cipher
3. Exit
Enter your choice: 1

Enter a Secret Message: sabin sir love you
Enter number of rails: 4

Encrypted Message: ssva ioeubnrl oi y
```



```
Encrypted Message: ssva ioeubnrl oi y

1. Encrypt using Rail Fence Cipher
2. Decrypt using Rail Fence Cipher
3. Exit
Enter your choice: 2

Enter the Ciphertext: ssva ioeubnrl oi y
Enter number of rails: 4

Decrypted Message: sabin sir love you
```

Encryption **Decryption**

5. CONCLUSION

In conclusion, the Monoalphabetic Cipher and Rail Fence Cipher are basic encryption techniques that help demonstrate the principles of cryptography. The Monoalphabetic Cipher substitutes each letter of the plaintext with a fixed letter, while the Rail Fence Cipher rearranges the text in a zigzag pattern. Both are simple to implement but not secure by modern standards, making them suitable for educational purposes rather than practical use.