

Lab-1

Task 1:

Write a program that takes an integer value K (i.e. shift value between +/- 26) and a plaintext message and returns the corresponding Caesar cipher. The program should also implement a decryption routine that reconstructs the original plaintext from the ciphertext.

Theory:

The Caesar cipher is a simple encryption technique that was used by Julius Caesar to send secret messages to his allies. It works by shifting the letters in the plaintext message by a certain number of positions, known as the “shift” or “key”.

The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

Thus, to cipher a given text we need an integer value, known as a shift which indicates the number of positions each letter of the text has been moved down.

The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1 ..., Z = 25. Encryption and Decryption of a letter by a shift n can be described mathematically as:

$$\text{Cipher } C = \text{Encryption (Plain Text, Key)} = (P + K) \bmod 26$$

$$\text{Plain Text } P = \text{Decryption (Cipher, Key)} = (C - K) \bmod 26$$

Source Code:

```
#include <iostream>
#include <string>

using namespace std;

string caesarEncrypt(int k, const string& plaintext)
{
    string ciphertext = "";
    for (char ch : plaintext)
    {
        if (isalpha(ch))
```

```

        {
            char base = isupper(ch) ? 'A' : 'a';
            ciphertext += static_cast<char>((ch - base + k + 26) % 26 +
base);
        }
        else
        {
            ciphertext += ch;
        }
    }
    return ciphertext;
}

string caesarDecrypt(int k, const string& ciphertext) {
    return caesarEncrypt(-k, ciphertext);
}

int main() {
    int shiftValue;
    string message;

    cout << "Enter the shift value (between +/- 26): ";
    cin >> shiftValue;

    if (shiftValue < -26 || shiftValue > 26) {
        cout << "Invalid shift value. Please enter a value between +/-
26." << endl;
        return 1;
    }

    cin.ignore();
    cout << "Enter the plaintext message: ";
    getline(cin, message);

    string encryptedMessage = caesarEncrypt(shiftValue, message);
    cout << "Encrypted message: " << encryptedMessage << endl;

    string decryptedMessage = caesarDecrypt(shiftValue, encryptedMessage);
    cout << "Decrypted message: " << decryptedMessage << endl;

    return 0;
}

```

Output:

```
D:\College\5th Semester\Cryp  ×  +  ∨  
Enter the shift value (between +/- 26): 3  
Enter the plaintext message: hello there  
Encrypted message: khood wkhuh  
Decrypted message: hello there  
  
-----  
Process exited after 6.959 seconds with return value 0  
Press any key to continue . . . |
```

Conclusion:

Hence, in this way we can implement encryption and decryption using Ceasar Cipher in the laboratory.

Task 2:

Write a program that asks user for key and plain text and displays the corresponding Vigenère cipher.

Theory:

Vigenère Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the Vigenère square or Vigenère table.

The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers. At different points in the encryption process, the cipher uses a different alphabet from one of the rows. The alphabet used at each point depends on a repeating keyword.

An easier implementation could be to visualize Vigenère algebraically by converting [A-Z] into numbers [0–25].

$$\text{Cipher } C_i = \text{Encryption (ith Plain Text, ith Key)} = (P_i + K_i) \bmod 26$$

$$\text{Plain Text } P_i = \text{Decryption (ith Cipher, ith Key)} = (C_i - K_i) \bmod 26$$

Source Code:

```
#include <iostream>
#include <string>

using namespace std;

string vigenereEncrypt(const string& key, const string& plaintext) {
    string ciphertext = "";
    int keyLength = key.length();
    int index = 0;

    for (char ch : plaintext) {
        if (isalpha(ch)) {
            char base = isupper(ch) ? 'A' : 'a';
            char shift = key[index % keyLength];
            shift = isupper(shift) ? shift - 'A' : shift - 'a';
            ciphertext += static_cast<char>((ch - base + shift + 26) % 26
+ base);
            index++;
        }
    }
}
```

```

        else {
            ciphertext += ch;
        }
    }
    return ciphertext;
}

int main() {
    string key, plaintext;

    cout << "Enter the key: ";
    cin >> key;

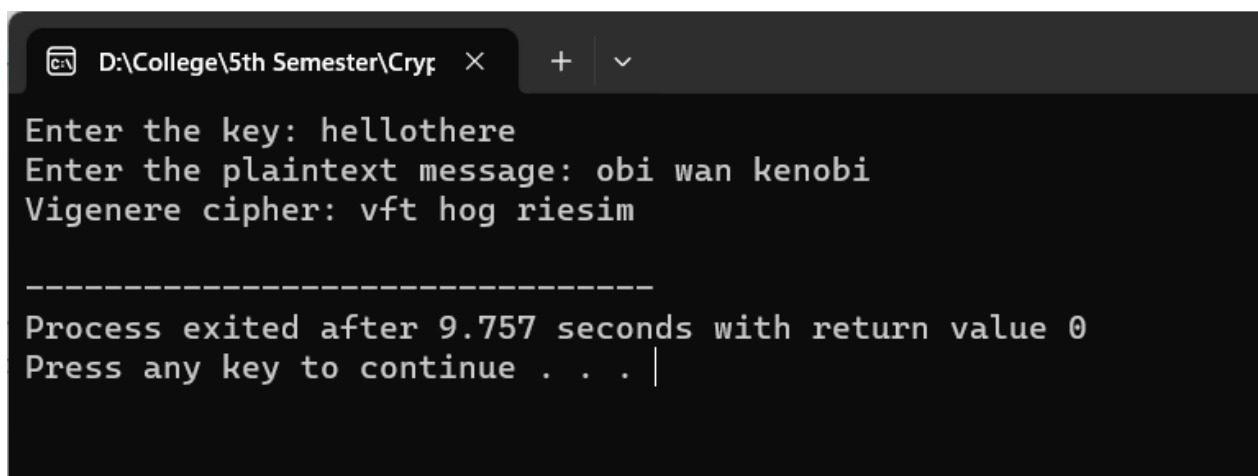
    cout << "Enter the plaintext message: ";
    cin.ignore();
    getline(cin, plaintext);

    string encryptedMessage = vigenereEncrypt(key, plaintext);
    cout << "Vigenere cipher: " << encryptedMessage << endl;

    return 0;
}

```

Output:



```

D:\College\5th Semester\Cryp X + v
Enter the key: hellothere
Enter the plaintext message: obi wan kenobi
Vigenere cipher: vft hog riesim

-----
Process exited after 9.757 seconds with return value 0
Press any key to continue . . . |

```

Conclusion:

Hence, in this way we can implement encryption and decryption using Vigenère Cipher in the laboratory.

Task 3:

Using the Rail Fence algorithm with depth 3, write a program to encrypt the message “I love my college”.

Theory:

The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded.

In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence. When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus, the alphabets of the message are written in a zig-zag manner. After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

Hence, rail matrix can be constructed accordingly. Once we've got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively. Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

Original Message: Hello World

| | | | | | | | | | | |
|---|---|---|---|---|--|---|--|---|--|---|
| H | | | | o | | | | r | | |
| | e | | l | | | o | | l | | |
| | | l | | | | W | | | | d |

Encrypted Message: Horel ollWd

Source Code:

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

void railFenceEncrypt(char* message, int depth) {
    int len = strlen(message);
    char fence[depth][len];
    char encryptedMessage[len];
    int row, col;
    int direction = 1;

    for (int i = 0; i < depth; i++) {
        for (int j = 0; j < len; j++) {
            fence[i][j] = '\\0';
        }
    }
}
```

```

row = 0;
col = 0;

for (int i = 0; i < len; i++) {
    fence[row][col] = message[i];
    if (row == 0) {
        direction = 1;
    }
    else if (row == depth - 1) {
        direction = -1;
    }
    row += direction;
    col++;
}

int index = 0;
for (int i = 0; i < depth; i++) {
    for (int j = 0; j < len; j++) {
        if (fence[i][j] != '\0') {
            encryptedMessage[index++] = fence[i][j];
        }
    }
}

encryptedMessage[len] = '\0';

printf("Encrypted message: %s\n", encryptedMessage);
}

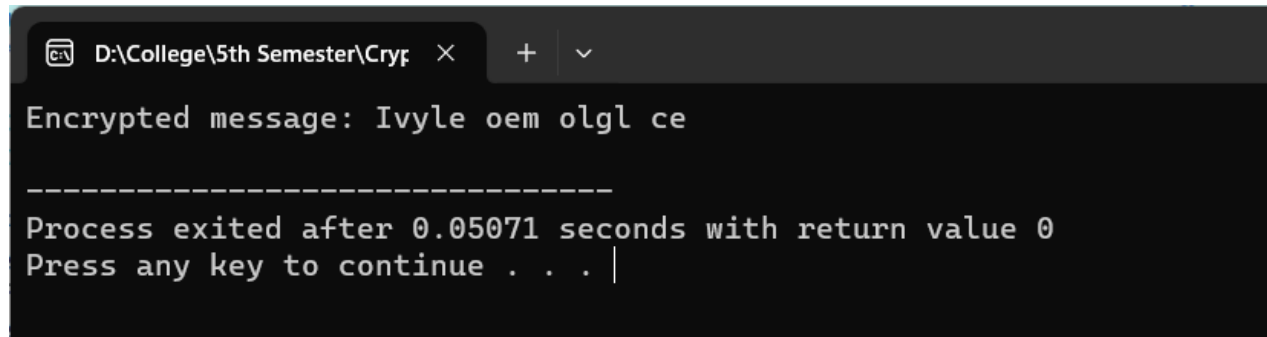
int main() {
    char message[] = "I love my college";
    int depth = 3;

    railFenceEncrypt(message, depth);

    return 0;
}

```

Output:



```
D:\College\5th Semester\Cryp  X  +  v
Encrypted message: Ivyle oem olgl ce
-----
Process exited after 0.05071 seconds with return value 0
Press any key to continue . . . |
```

Conclusion:

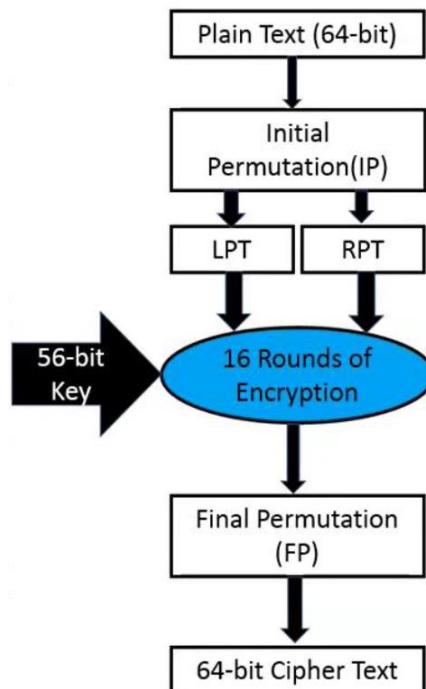
Hence, in this way we can implement encryption of a given plaintext using Rail Fence Cipher in the laboratory.

Task 4:

Write a program to demonstrate the calculation of initial permutation of a plain text in DES algorithm.

Theory:

Data Encryption Standard (DES) is a block cipher with a 56-bit key length that has played a significant role in data security. Data encryption standard (DES) has been found vulnerable to very powerful attacks therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.



As we can see the first step in DES is the Initial Permutation. As we have noted, the initial permutation (IP) happens only once and it happens before the first round. It suggests how the transposition in IP should proceed, as shown in the figure. For example, it says that the IP replaces the first bit of the original plain text block with the 58th bit of the original plain text, the second bit with the 50th bit of the original plain text block, and so on.

This is nothing but jugglery of bit positions of the original plain text block. the same rule applies to all the other bit positions shown in the figure.

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|---|----|----|----|----|----|----|----|---|
| 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 | 33 | 45 | 37 | 29 | 21 | 13 | 5 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

Figure - Initial permutation table

Source Code:

```
#include <stdio.h>

// Initial Permutation (IP) table
int initial_permutation_table[] = {
    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
};

void initial_permutation(char* plain_text, char* initial_permuted_text) {
    for (int i = 0; i < 64; i++) {
        int bit_position = initial_permutation_table[i] - 1;
        int byte_position = bit_position / 8;
        int bit_offset = 7 - (bit_position % 8);

        // Extract the bit from the byte
        char bit = (plain_text[byte_position] >> bit_offset) & 1;

        // Set the corresponding bit in the initial_permuted_text
        initial_permuted_text[i / 8] |= (bit << (7 - (i % 8)));
    }
}

int main() {
    char plain_text[] = "Hello DES";
    char initial_permuted_text[8] = {0}; // Initialize with zeros

    initial_permutation(plain_text, initial_permuted_text);

    printf("Plaintext: %s\n", plain_text);
}
```

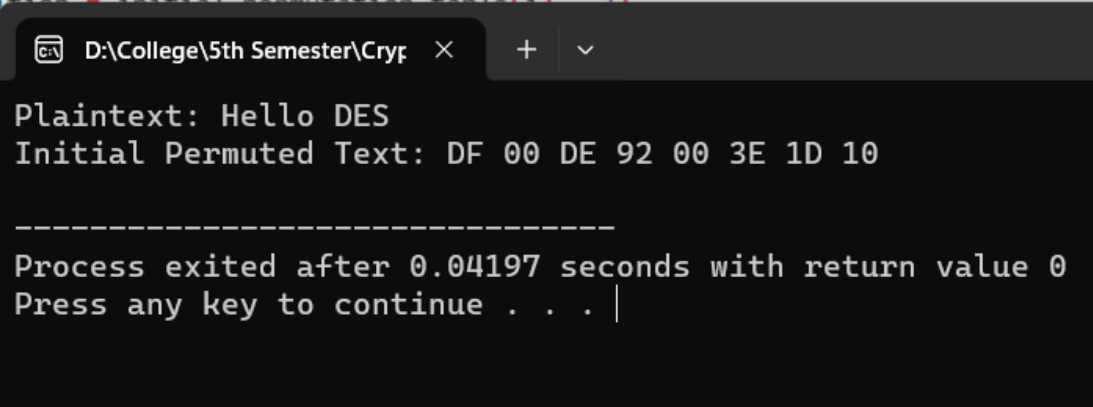
```

printf("Initial Permuted Text: ");
for (int i = 0; i < 8; i++) {
    printf("%02X ", (unsigned char)initial_permuted_text[i]);
}
printf("\n");

return 0;
}

```

Output:



```

D:\College\5th Semester\Cryp × + v
Plaintext: Hello DES
Initial Permuted Text: DF 00 DE 92 00 3E 1D 10

-----
Process exited after 0.04197 seconds with return value 0
Press any key to continue . . . |

```

Conclusion:

Hence, in this way we can implement the Initial Permutation Step of DES Encryption Algorithm in the laboratory.