**Lab 1**

**Lab 1.1: Write a program that takes an integer value K (i.e. shift value between +/- 26) and a plaintext message and returns the corresponding Ceasar cipher. The program should also implement a decryption routine that reconstructs the original plaintext from the ciphertext.**

**Algorithm:**

Ceasar cipher:

1. Choose a shift value between 1 and 25.
2. Write down the alphabet in order from A to Z.
3. Create a new alphabet by shifting each letter of the original alphabet by the shift value.
4. Replace each letter of the message with the corresponding letter from the new alphabet.
5. To decrypt the message, shift each letter back by the same amount.

**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 shift value (1 - 26): ";

cin >> shiftValue;

if (shiftValue < -26 || shiftValue > 26) {

cout << "Invalid shift value. Enter a value between 1 to 26." << endl;

return 1;

}

cin.ignore();

cout << "Enter plaintext: ";

getline(cin, message);

string encryptedMessage = caesarEncrypt(shiftValue, message);

cout << "Cipher text: " << encryptedMessage << endl;

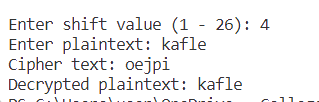
string decryptedMessage = caesarDecrypt(shiftValue, encryptedMessage);

cout << "Decrypted plaintext: " << decryptedMessage << endl;

return 0;

}

**Output:**



**Lab 1.2: Write a program that asks user for key and plain text and displays the corresponding Vigenere cipher.**

**Algorithm:**

Vigenere cipher:

Encryption:

1. Repeat the key to match the length of the plaintext.
2. For each letter in the plaintext:
   1. Find the corresponding row in the Vigenere table based on the plaintext letter.
   2. Find the column in that row based on the key letter.
   3. The letter at the intersection of that row and column is the ciphertext letter.

Decryption:

1. Repeat the key to match the length of the ciphertext.
2. For each letter in the ciphertext:
   1. Find the corresponding row in the Vigenere table based on the ciphertext letter.
   2. Find the key letter that, when used in the corresponding row, results in the plaintext letter.
   3. That key letter is the decryption key for that ciphertext letter.

**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 plaintext: ";

cin.ignore();

getline(cin, plaintext);

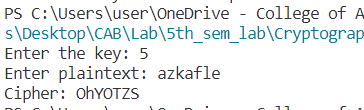
string encryptedMessage = vigenereEncrypt(key, plaintext);

cout << "Cipher: " << encryptedMessage << endl;

return 0;

}

**Output:**



**Lab 1.3: Using the Rail Fence algorithm with depth 3, write a program to encrypt the message “I love my college”.**

**Algorithm:**

Rail Fence:

Encryption:

1. Choose the number of rails (key).
2. Write the plaintext message on the rails in a zigzag pattern, starting from the top rail and alternating downward.
3. Read the ciphertext off row by row, ignoring empty spaces.

Decryption:

1. Know the number of rails (key)
2. Fill empty rails with spaces to match the original message length.
3. Write the ciphertext letters on the rails in a zigzag pattern, starting from the top rail and alternating downward.
4. Read the plaintext off row by row, ignoring empty spaces.

**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:**



**Lab 1.4: Write a program to demonstrate the calculation of initial permutation of a plain text in DES algorithm.**

**Algorithm:**

Data Encryption Standard:

1. First provide 64-bit plain text as input.
2. Then initialize initial permutation (IP) table.
   1. Create a lookup table (IP table) with 64 elements, where each element holds a unique integer value from 1 to 64, representing the new position of a corresponding bit in the permuted output.
   2. The specific values in the IP table are fixed as defined in the DES specification.
3. Iterate through each bit of the input plaintext block:
   1. Retrieve the current bit's position in the block (index).
   2. Use the IP table to lookup the new position for this bit.
   3. Place the bit at the corresponding new position in a new 64-bit output block.
4. Return the 64-bit output block as the permuted plaintext.

**Source Code:**

#include <stdio.h>

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);

char bit = (plain\_text[byte\_position] >> bit\_offset) & 1;

initial\_permuted\_text[i / 8] |= (bit << (7 - (i % 8)));

}

}

int main() {

char plain\_text[] = "kafleaz9098";

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:**

