Practical :- 1

#include <iostream>

#include <string>

// Function to encrypt the text using Caesar Cipher

std::string encryptCaesarCipher(std::string text, int shift) {

std::string result = "";

// Traverse text

for (int i = 0; i < text.length(); i++) {

char ch = text[i];

// Encrypt uppercase letters

if (isupper(ch))

result += char(int(ch + shift - 65) % 26 + 65);

// Encrypt lowercase letters

else if (islower(ch))

result += char(int(ch + shift - 97) % 26 + 97);

// Encrypt digits

else if (isdigit(ch))

result += char(int(ch + shift - 48) % 10 + 48);

// Leave other characters unchanged

else

result += ch;

}

return result;

}

// Function to decrypt the text using Caesar Cipher

std::string decryptCaesarCipher(std::string text, int shift) {

std::string result = "";

// Traverse text

for (int i = 0; i < text.length(); i++) {

char ch = text[i];

// Decrypt uppercase letters

if (isupper(ch))

result += char(int(ch - shift - 65 + 26) % 26 + 65);

// Decrypt lowercase letters

else if (islower(ch))

result += char(int(ch - shift - 97 + 26) % 26 + 97);

// Decrypt digits

else if (isdigit(ch))

result += char(int(ch - shift - 48 + 10) % 10 + 48);

// Leave other characters unchanged

else

result += ch;

}

return result;

}

int main() {

std::string text;

int shift;

char choice;

std::cout << "Enter the text: ";

std::getline(std::cin, text);

std::cout << "Enter the shift value: ";

std::cin >> shift;

std::cout << "Do you want to (e)ncrypt or (d)ecrypt? ";

std::cin >> choice;

if (choice == 'e' || choice == 'E') {

std::cout << "Encrypted text: " << encryptCaesarCipher(text, shift) << std::endl;

} else if (choice == 'd' || choice == 'D') {

std::cout << "Decrypted text: " << decryptCaesarCipher(text, shift) << std::endl;

} else {

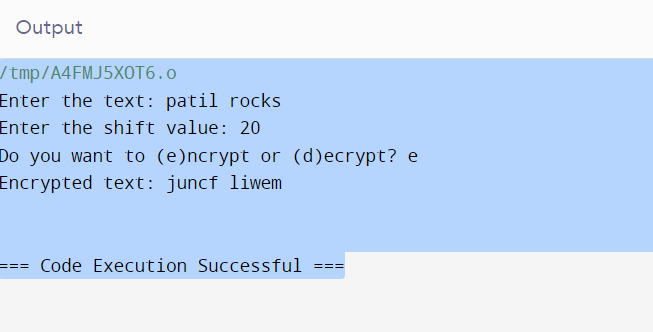
std::cout << "Invalid choice" << std::endl;

}

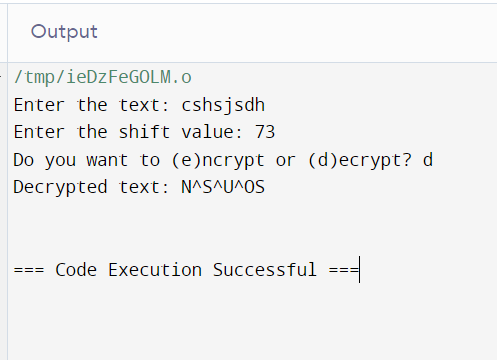
return 0;

}

**Encryption :-**



**Decryption :-**

****

Practical No :- 2

#include <iostream>

#include <unordered\_map>

#include <string>

// Function to generate the encryption and decryption maps based on the key

void generateMaps(std::string key, std::unordered\_map<char, char>& encryptMap, std::unordered\_map<char, char>& decryptMap) {

std::string alphabet = "abcdefghijklmnopqrstuvwxyz";

for (int i = 0; i < alphabet.length(); i++) {

encryptMap[alphabet[i]] = key[i];

decryptMap[key[i]] = alphabet[i];

}

}

// Function to encrypt the text using Monoalphabetic Cipher

std::string encryptMonoalphabeticCipher(std::string text, std::unordered\_map<char, char>& encryptMap) {

std::string result = "";

for (char ch : text) {

if (isalpha(ch)) {

char lower = tolower(ch);

result += isupper(ch) ? toupper(encryptMap[lower]) : encryptMap[lower];

} else {

result += ch;

}

}

return result;

}

// Function to decrypt the text using Monoalphabetic Cipher

std::string decryptMonoalphabeticCipher(std::string text, std::unordered\_map<char, char>& decryptMap) {

std::string result = "";

for (char ch : text) {

if (isalpha(ch)) {

char lower = tolower(ch);

result += isupper(ch) ? toupper(decryptMap[lower]) : decryptMap[lower];

} else {

result += ch;

}

}

return result;

}

int main() {

std::string text, key;

char choice;

std::unordered\_map<char, char> encryptMap, decryptMap;

// Prompt for key

std::cout << "Enter the 26-letter key for the cipher (e.g., QWERTYUIOPASDFGHJKLZXCVBNM): ";

std::cin >> key;

// Generate maps

generateMaps(key, encryptMap, decryptMap);

// Clear the input buffer

std::cin.ignore();

// Prompt for text and choice

std::cout << "Enter the text: ";

std::getline(std::cin, text);

std::cout << "Do you want to (e)ncrypt or (d)ecrypt? ";

std::cin >> choice;

if (choice == 'e' || choice == 'E') {

std::cout << "Encrypted text: " << encryptMonoalphabeticCipher(text, encryptMap) << std::endl;

} else if (choice == 'd' || choice == 'D') {

std::cout << "Decrypted text: " << decryptMonoalphabeticCipher(text, decryptMap) << std::endl;

} else {

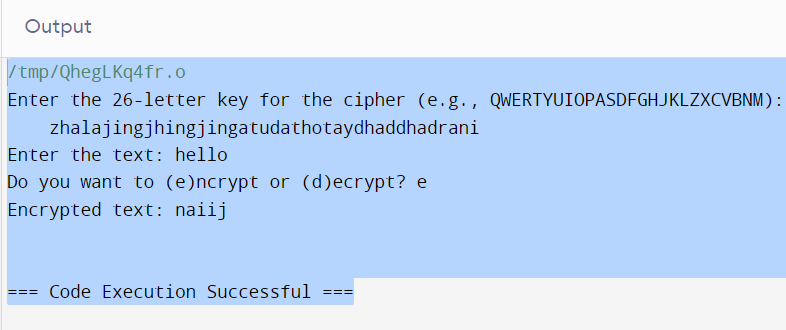
std::cout << "Invalid choice" << std::endl;

}

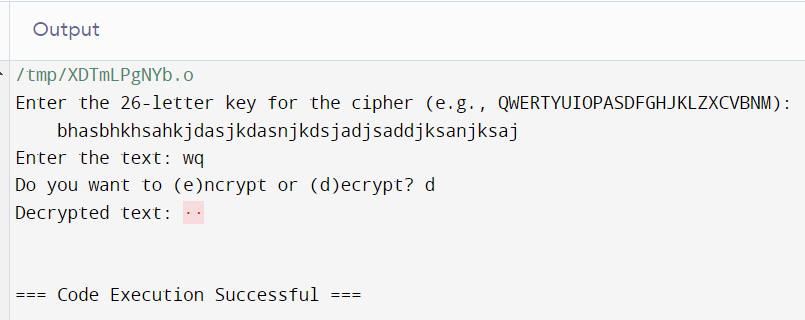
return 0;

}

**Encryption :-**

****

**Decryption :-**

****

**Practical No :- 3**

**#include <iostream>**

**#include <vector>**

**#include <algorithm>**

**#include <cctype>**

**#include <string>**

**#include <unordered\_set>**

**// Function to generate the Playfair matrix based on the key**

**void generateMatrix(std::string key, char matrix[5][5]) {**

**std::string keyString = "";**

**std::unordered\_set<char> usedChars;**

**// Add key characters to keyString, removing duplicates and ignoring 'J'**

**for (char ch : key) {**

**ch = toupper(ch);**

**if (ch == 'J') ch = 'I';**

**if (usedChars.find(ch) == usedChars.end() && isalpha(ch)) {**

**keyString += ch;**

**usedChars.insert(ch);**

**}**

**}**

**// Add remaining letters to keyString**

**for (char ch = 'A'; ch <= 'Z'; ch++) {**

**if (ch == 'J') continue;**

**if (usedChars.find(ch) == usedChars.end()) {**

**keyString += ch;**

**usedChars.insert(ch);**

**}**

**}**

**// Fill the matrix**

**int k = 0;**

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

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

**matrix[i][j] = keyString[k++];**

**}**

**}**

**}**

**// Function to find the position of a character in the matrix**

**void findPosition(char matrix[5][5], char ch, int &row, int &col) {**

**if (ch == 'J') ch = 'I'; // Treat 'J' as 'I'**

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

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

**if (matrix[i][j] == ch) {**

**row = i;**

**col = j;**

**return;**

**}**

**}**

**}**

**}**

**// Function to process text by removing non-alphabetic characters and handling duplicate letters in digraphs**

**std::string processText(std::string text) {**

**std::string result = "";**

**for (char ch : text) {**

**if (isalpha(ch)) {**

**ch = toupper(ch);**

**result += (ch == 'J') ? 'I' : ch;**

**}**

**}**

**// Handle duplicate letters in digraphs**

**for (size\_t i = 0; i < result.length(); i += 2) {**

**if (i + 1 < result.length() && result[i] == result[i + 1]) {**

**result.insert(i + 1, "X");**

**}**

**}**

**// If the processed text has an odd number of characters, add 'X' at the end**

**if (result.length() % 2 != 0) {**

**result += 'X';**

**}**

**return result;**

**}**

**// Function to encrypt the text using Playfair Cipher**

**std::string encryptPlayfairCipher(std::string text, char matrix[5][5]) {**

**std::string result = "";**

**text = processText(text);**

**for (size\_t i = 0; i < text.length(); i += 2) {**

**char first = text[i];**

**char second = text[i + 1];**

**int row1, col1, row2, col2;**

**findPosition(matrix, first, row1, col1);**

**findPosition(matrix, second, row2, col2);**

**if (row1 == row2) {**

**result += matrix[row1][(col1 + 1) % 5];**

**result += matrix[row2][(col2 + 1) % 5];**

**} else if (col1 == col2) {**

**result += matrix[(row1 + 1) % 5][col1];**

**result += matrix[(row2 + 1) % 5][col2];**

**} else {**

**result += matrix[row1][col2];**

**result += matrix[row2][col1];**

**}**

**}**

**return result;**

**}**

**// Function to decrypt the text using Playfair Cipher**

**std::string decryptPlayfairCipher(std::string text, char matrix[5][5]) {**

**std::string result = "";**

**text = processText(text);**

**for (size\_t i = 0; i < text.length(); i += 2) {**

**char first = text[i];**

**char second = text[i + 1];**

**int row1, col1, row2, col2;**

**findPosition(matrix, first, row1, col1);**

**findPosition(matrix, second, row2, col2);**

**if (row1 == row2) {**

**result += matrix[row1][(col1 + 4) % 5];**

**result += matrix[row2][(col2 + 4) % 5];**

**} else if (col1 == col2) {**

**result += matrix[(row1 + 4) % 5][col1];**

**result += matrix[(row2 + 4) % 5][col2];**

**} else {**

**result += matrix[row1][col2];**

**result += matrix[row2][col1];**

**}**

**}**

**return result;**

**}**

**int main() {**

**std::string text, key;**

**char choice;**

**char matrix[5][5];**

**// Prompt for key**

**std::cout << "Enter the key for the cipher: ";**

**std::getline(std::cin, key);**

**// Generate matrix**

**generateMatrix(key, matrix);**

**// Prompt for text and choice**

**std::cout << "Enter the text: ";**

**std::getline(std::cin, text);**

**std::cout << "Do you want to (e)ncrypt or (d)ecrypt? ";**

**std::cin >> choice;**

**if (choice == 'e' || choice == 'E') {**

**std::cout << "Encrypted text: " << encryptPlayfairCipher(text, matrix) << std::endl;**

**} else if (choice == 'd' || choice == 'D') {**

**std::cout << "Decrypted text: " << decryptPlayfairCipher(text, matrix) << std::endl;**

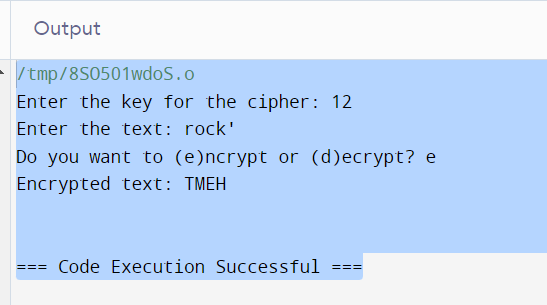
**} else {**

**std::cout << "Invalid choice" << std::endl;**

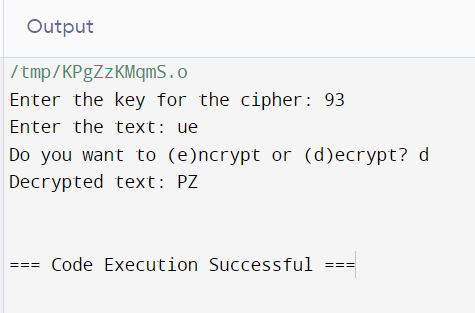
**}**

**return 0;**

**Encryption :-**

****

**Decryption :-**

****

**Practical No:- 4**

**#include <iostream>**

**#include <string>**

**// Function to extend the key to match the length of the text**

**std::string extendKey(const std::string &text, const std::string &key) {**

**std::string extendedKey = key;**

**int textLength = text.length();**

**int keyLength = key.length();**

**for (int i = 0; i < textLength - keyLength; i++) {**

**extendedKey += key[i % keyLength];**

**}**

**return extendedKey;**

**}**

**// Function to encrypt the text using Vigenère Cipher**

**std::string encryptVigenereCipher(const std::string &text, const std::string &key) {**

**std::string encryptedText = "";**

**std::string extendedKey = extendKey(text, key);**

**for (size\_t i = 0; i < text.length(); i++) {**

**char ch = text[i];**

**if (isalpha(ch)) {**

**char base = isupper(ch) ? 'A' : 'a';**

**char keyCh = toupper(extendedKey[i]) - 'A';**

**encryptedText += (ch - base + keyCh) % 26 + base;**

**} else {**

**encryptedText += ch;**

**}**

**}**

**return encryptedText;**

**}**

**// Function to decrypt the text using Vigenère Cipher**

**std::string decryptVigenereCipher(const std::string &text, const std::string &key) {**

**std::string decryptedText = "";**

**std::string extendedKey = extendKey(text, key);**

**for (size\_t i = 0; i < text.length(); i++) {**

**char ch = text[i];**

**if (isalpha(ch)) {**

**char base = isupper(ch) ? 'A' : 'a';**

**char keyCh = toupper(extendedKey[i]) - 'A';**

**decryptedText += (ch - base - keyCh + 26) % 26 + base;**

**} else {**

**decryptedText += ch;**

**}**

**}**

**return decryptedText;**

**}**

**int main() {**

**std::string text, key;**

**char choice;**

**// Prompt for key**

**std::cout << "Enter the key for the cipher: ";**

**std::getline(std::cin, key);**

**// Prompt for text and choice**

**std::cout << "Enter the text: ";**

**std::getline(std::cin, text);**

**std::cout << "Do you want to (e)ncrypt or (d)ecrypt? ";**

**std::cin >> choice;**

**if (choice == 'e' || choice == 'E') {**

**std::cout << "Encrypted text: " << encryptVigenereCipher(text, key) << std::endl;**

**} else if (choice == 'd' || choice == 'D') {**

**std::cout << "Decrypted text: " << decryptVigenereCipher(text, key) << std::endl;**

**} else {**

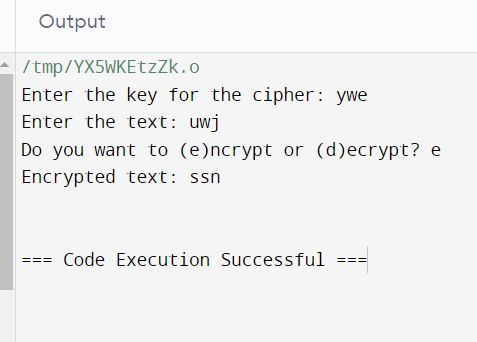
**std::cout << "Invalid choice" << std::endl;**

**}**

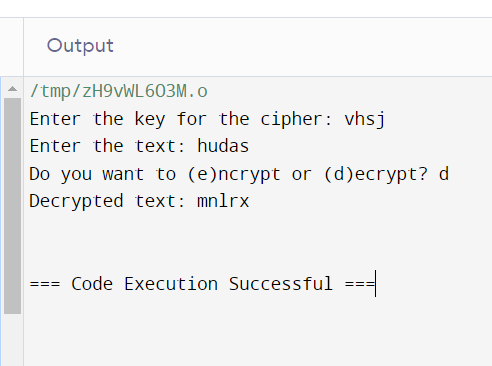
**return 0;**

**}**

**Encryption:-**

****

**Decryption:-**

****