CRYPTOGRAPHY AND NETWORK SECURITY
LAB - 3

Name of the Student: SreeDananjay S
Registration Number: 21BAI1807
Slot: L31+L32
Course Code: BCSE309P
Programme: Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning
School: School of Computer Science and Engineering(SCOPE)

Q2)

a) Hill Cipher algorithm implementation in Java

Code:

```
import java.util.ArrayList;
import java.util.Scanner;
public class HillCipher{
  //method to accept key matrix
  private static int[][] getKeyMatrix() {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter key matrix:");
    String key = sc.nextLine();
    //int len = key.length();
    double sq = Math.sqrt(key.length());
    if (sq != (long) sq) {
      System.out.println("Cannot Form a square matrix");
    }
    int len = (int) sq;
    int[][] keyMatrix = new int[len][len];
    int k = 0;
    for (int i = 0; i < len; i++)
    {
      for (int j = 0; j < len; j++)
      {
        keyMatrix[i][j] = ((int) key.charAt(k)) - 97;
        k++;
      }
    }
    return keyMatrix;
  }
  // Below method checks whether the key matrix is valid (det=0)
  private static void isValidMatrix(int[][] keyMatrix) {
    int det = keyMatrix[0][0] * keyMatrix[1][1] - keyMatrix[0][1] * keyMatrix[1][0];
    // If det=0, throw exception and terminate
    if(det == 0) {
      throw new java.lang.Error("Det equals to zero, invalid key matrix!");
```

```
    }
  }
  // This method checks if the reverse key matrix is valid (matrix mod26 = (1,0,0,1)
    private static void isValidReverseMatrix(int[][] keyMatrix, int[][] reverseMatrix) {
    int[][] product = new int[2][2];
    // Find the product matrix of key matrix times reverse key matrix
    product[0][0] = (keyMatrix[0][0]*reverseMatrix[0][0] + keyMatrix[0][1] *
reverseMatrix[1][0]) % 26;
    product[0][1] = (keyMatrix[0][0]*reverseMatrix[0][1] + keyMatrix[0][1] *
reverseMatrix[1][1]) % 26;
    product[1][0] = (keyMatrix[1][0]*reverseMatrix[0][0] + keyMatrix[1][1] *
reverseMatrix[1][0]) % 26;
    product[1][1] = (keyMatrix[1][0]*reverseMatrix[0][1] + keyMatrix[1][1] *
reverseMatrix[1][1]) % 26;
    // Check if a=1 and b=0 and c=0 and d=1
    // If not, throw exception and terminate
    if(product[0][0] != 1 || product[0][1] != 0 || product[1][0] != 0 || product[1][1] != 1) {
      throw new java.lang.Error("Invalid reverse matrix found!");
    }
  }
  // This method calculates the reverse key matrix
  private static int[][] reverseMatrix(int[][] keyMatrix) {
    int detmod26 = (keyMatrix[0][0] * keyMatrix[1][1] - keyMatrix[0][1] *
keyMatrix[1][0]) % 26; // Calc det
    int factor;
    int[][] reverseMatrix = new int[2][2];
    // Find the factor for which is true that
    // factor*det = 1 mod 26
    for(factor=1; factor < 26; factor++)
    {
      if((detmod26 * factor) % 26 == 1)
      {
        break;
      }
    }
    // Calculate the reverse key matrix elements using the factor found
    reverseMatrix[0][0] = keyMatrix[1][1]        * factor % 26;
    reverseMatrix[0][1] = (26 - keyMatrix[0][1])   * factor % 26;
```

```
    reverseMatrix[1][0] = (26 - keyMatrix[1][0])    * factor % 26;
    reverseMatrix[1][1] = keyMatrix[0][0]         * factor % 26;
    return reverseMatrix;
  }
  // This method echoes the result of encrypt/decrypt
  private static void echoResult(String label, int adder, ArrayList<Integer> phrase) {
    int i;
    System.out.print(label);
    // Loop for each pair
    for(i=0; i < phrase.size(); i += 2) {
      System.out.print(Character.toChars(phrase.get(i) + (64 + adder)));
      System.out.print(Character.toChars(phrase.get(i+1) + (64 + adder)));
      if(i+2 <phrase.size()) {
        System.out.print("-");
      }
    }
    System.out.println();
  }
  // This method makes the actual encryption
  public static void encrypt(String phrase, boolean alphaZero)
  {
    int i;
    int adder = alphaZero ? 1 : 0; // For calclulations depending on the alphabet
    int[][] keyMatrix;
    ArrayList<Integer> phraseToNum = new ArrayList<>();
    ArrayList<Integer> phraseEncoded = new ArrayList<>();
    // Delete all non-english characters, and convert phrase to upper case
    phrase = phrase.replaceAll("[^a-zA-Z]","").toUpperCase();

    // If phrase length is not an even number, add "Q" to make it even
    if(phrase.length() % 2 == 1) {
      phrase += "Q";
    }
    // Get the 2x2 key matrix from sc
    keyMatrix = getKeyMatrix();
    // Check if the matrix is valid (det != 0)
    isValidMatrix(keyMatrix);
    // Convert characters to numbers according to their
```

```
    // place in ASCII table minus 64 positions (A=65 in ASCII table)
    // If we use A=0 alphabet, subtract one more (adder)
    for(i=0; i < phrase.length(); i++) {
      phraseToNum.add(phrase.charAt(i) - (64 + adder));
    }
    // Find the product per pair of the phrase with the key matrix modulo 26
    // If we use A=1 alphabet and result is 0, replace it with 26 (Z)
    for(i=0; i < phraseToNum.size(); i += 2) {
      int x = (keyMatrix[0][0] * phraseToNum.get(i) + keyMatrix[0][1] *
phraseToNum.get(i+1)) % 26;
      int y = (keyMatrix[1][0] * phraseToNum.get(i) + keyMatrix[1][1] *
phraseToNum.get(i+1)) % 26;
      phraseEncoded.add(alphaZero ? x : (x == 0 ? 26 : x ));
      phraseEncoded.add(alphaZero ? y : (y == 0 ? 26 : y ));
    }
    // Print the result
    echoResult("Encoded phrase: ", adder, phraseEncoded);
  }
  // This method makes the actual decryption
  public static void decrypt(String phrase, boolean alphaZero)
  {
    int i, adder = alphaZero ? 1 : 0;
    int[][] keyMatrix, revKeyMatrix;
    ArrayList<Integer> phraseToNum = new ArrayList<>();
    ArrayList<Integer> phraseDecoded = new ArrayList<>();
    // Delete all non-english characters, and convert phrase to upper case
    phrase = phrase.replaceAll("[^a-zA-Z]","").toUpperCase();

    // Get the 2x2 key matrix from sc
    keyMatrix = getKeyMatrix();
    // Check if the matrix is valid (det != 0)
    isValidMatrix(keyMatrix);
    // Convert numbers to characters according to their
    // place in ASCII table minus 64 positions (A=65 in ASCII table)
    // If we use A=0 alphabet, subtract one more (adder)
    for(i=0; i < phrase.length(); i++) {
      phraseToNum.add(phrase.charAt(i) - (64 + adder));
    }
```

```
    // Find the reverse key matrix
    revKeyMatrix = reverseMatrix(keyMatrix);
    // Check if the reverse key matrix is valid (product = 1,0,0,1)
    isValidReverseMatrix(keyMatrix, revKeyMatrix);
    // Find the product per pair of the phrase with the reverse key matrix modulo 26
    for(i=0; i < phraseToNum.size(); i += 2) {
      phraseDecoded.add((revKeyMatrix[0][0] * phraseToNum.get(i) +
revKeyMatrix[0][1] * phraseToNum.get(i+1)) % 26);
      phraseDecoded.add((revKeyMatrix[1][0] * phraseToNum.get(i) +
revKeyMatrix[1][1] * phraseToNum.get(i+1)) % 26);
    }
    // Print the result
    echoResult("Decoded phrase: ", adder, phraseDecoded);
  }
  //main method
  public static void main(String[] args) {
    String opt, phrase;
    byte[] p;
    Scanner sc = new Scanner(System.in);
    System.out.println("Hill Cipher Implementation (2x2)");
    System.out.println("-------------------------");
    System.out.println("1. Encrypt text (A=0,B=1,...Z=25)");
    System.out.println("2. Decrypt text (A=0,B=1,...Z=25)");
    System.out.println("3. Encrypt text (A=1,B=2,...Z=26)");
    System.out.println("4. Decrypt text (A=1,B=2,...Z=26)");
    System.out.println();
    System.out.println("Type any other character to exit");
    System.out.println();
    System.out.print("Select your choice: ");
    opt = sc.nextLine();
    switch (opt)
    {
      case "1":
        System.out.print("Enter phrase to encrypt: ");
        phrase = sc.nextLine();
        encrypt(phrase, true);
        break;
      case "2":
```

```
                System.out.print("Enter phrase to decrypt: ");
                phrase = sc.nextLine();
                decrypt(phrase, true);
                break;
            case "3":
                System.out.print("Enter phrase to encrypt: ");
                phrase = sc.nextLine();
                encrypt(phrase, false);
                break;
            case "4":
                System.out.print("Enter phrase to decrypt: ");
                phrase = sc.nextLine();
                decrypt(phrase, false);
                break;
        }
      }
    }
```

Sample output Screenshot:

```
➜  Lab3 javac HillCipher.java
➜  Lab3 java HillCipher
Hill Cipher Implementation (2x2)
------------------------
1. Encrypt text (A=0,B=1,...Z=25)
2. Decrypt text (A=0,B=1,...Z=25)
3. Encrypt text (A=1,B=2,...Z=26)
4. Decrypt text (A=1,B=2,...Z=26)

Type any other character to exit

Select your choice: 1
Enter phrase to encrypt: Dananjay
Enter key matrix:
nwkc
Encoded phrase: NE-NA-DS-IW
Decrypted text is : Dananjay%
```

b) Viginere Cipher Implementation in C

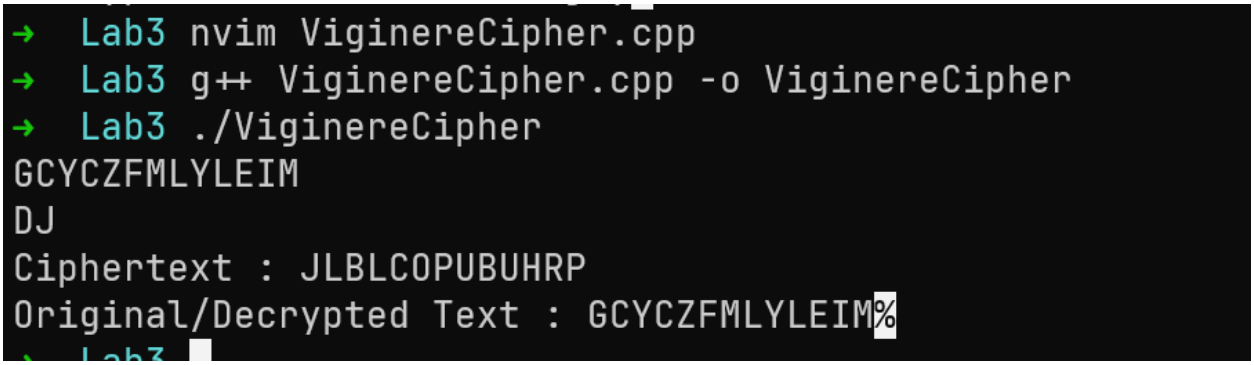
Code:

```
// C++ code to implement Vigenere Cipher
#include <bits/stdc++.h>
using namespace std;

// This function generates the key in
// a cyclic manner until it's length isn't
// equal to the length of original text
string generateKey(string str, string key)
{
   int x = str.size();

   for (int i = 0;; i++) {
      if (x == i)
         i = 0;
      if (key.size() == str.size())
         break;
      key.push_back(key[i]);
   }
   return key;
}

// This function returns the encrypted text
// generated with the help of the key
string cipherText(string str, string key)
{
   string cipher_text;

   for (int i = 0; i < str.size(); i++) {
      // converting in range 0-25
      char x = (str[i] + key[i]) % 26;

      // convert into alphabets(ASCII)
      x += 'A';

      cipher_text.push_back(x);
   }
   return cipher_text;
```

```
}

// This function decrypts the encrypted text
// and returns the original text
string originalText(string cipher_text, string key)
{
   string orig_text;

   for (int i = 0; i < cipher_text.size(); i++) {
      // converting in range 0-25
      char x = (cipher_text[i] - key[i] + 26) % 26;

      // convert into alphabets(ASCII)
      x += 'A';
      orig_text.push_back(x);
   }
   return orig_text;
}

// Driver program to test the above function
int main()
{
   string str,keyword;
   cin >> str >> keyword;
   if (any_of(str.begin(), str.end(), ::islower))
      transform(str.begin(), str.end(), str.begin(),
            ::toupper);
   if (any_of(keyword.begin(), keyword.end(), ::islower))
      transform(keyword.begin(), keyword.end(),
            keyword.begin(), ::toupper);

   string key = generateKey(str, keyword);
   string cipher_text = cipherText(str, key);

   cout << "Ciphertext : " << cipher_text << "\n";

   cout << "Original/Decrypted Text : "
      << originalText(cipher_text, key);
```

```
    return 0;
}
```

Output Screenshots:



Q3) Rail Fence Cipher:

Code:

```
#include <bits/stdc++.h>

using namespace std;


// function to encrypt a message

string encryptRailFence(string text, int key)

{

// create the matrix to cipher plain text

// key = rows , length(text) = columns

char rail[key][(text.length())];


// filling the rail matrix to distinguish filled

// spaces from blank ones

for (int i=0; i < key; i++)
```

```
for (int j = 0; j < text.length(); j++)
rail[i][j] = '\n';

// to find the direction
bool dir_down = false;
int row = 0, col = 0;

for (int i=0; i < text.length(); i++)
{
// check the direction of flow
// reverse the direction if we've just
// filled the top or bottom rail
if (row == 0 || row == key-1)
dir_down = !dir_down;

// fill the corresponding alphabet
rail[row][col++] = text[i];

// find the next row using direction flag
dir_down?row++ : row--;
}

//now we can construct the cipher using the rail matrix
string result;
for (int i=0; i < key; i++)
for (int j=0; j < text.length(); j++)
```

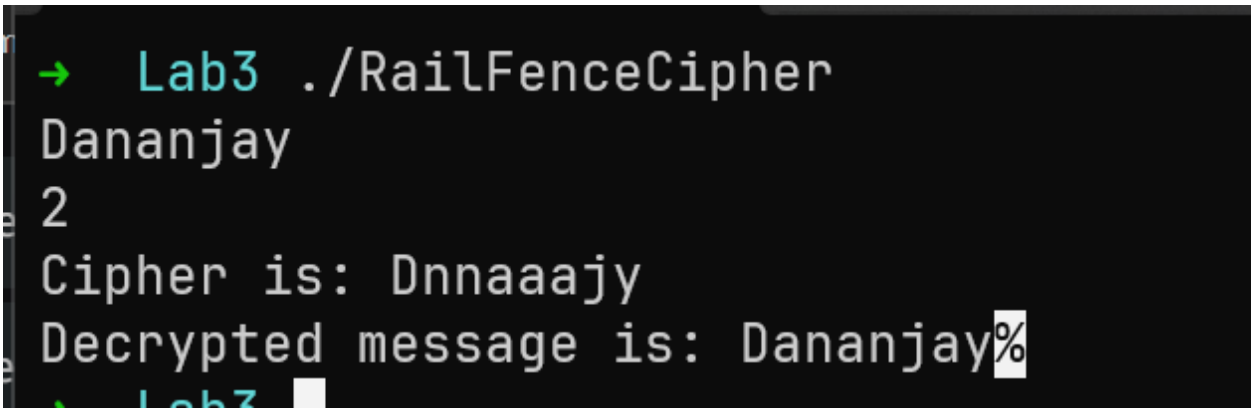
```
if (rail[i][j]!='\n')
result.push_back(rail[i][j]);

return result;
}

// This function receives cipher-text and key
// and returns the original text after decryption
string decryptRailFence(string cipher, int key)
{
// create the matrix to cipher plain text
// key = rows , length(text) = columns
char rail[key][cipher.length()];

// filling the rail matrix to distinguish filled
// spaces from blank ones
for (int i=0; i < key; i++)
for (int j=0; j < cipher.length(); j++)
rail[i][j] = '\n';

// to find the direction
bool dir_down;

int row = 0, col = 0;

// mark the places with '*'
```

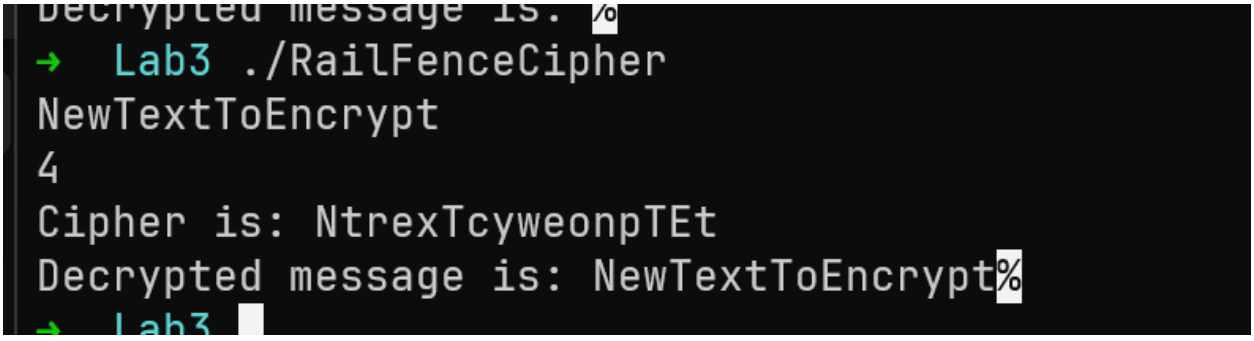
```
for (int i=0; i < cipher.length(); i++)
{
// check the direction of flow
if (row == 0)
dir_down = true;
if (row == key-1)
dir_down = false;

// place the marker
rail[row][col++] = '*';

// find the next row using direction flag
dir_down?row++ : row--;
}

// now we can construct the fill the rail matrix
int index = 0;
for (int i=0; i<key; i++)
for (int j=0; j<cipher.length(); j++)
if (rail[i][j] == '*' && index<cipher.length())
rail[i][j] = cipher[index++];


// now read the matrix in zig-zag manner to construct
// the resultant text
string result;
```

```
row = 0, col = 0;
for (int i=0; i< cipher.length(); i++)
{
// check the direction of flow
if (row == 0)
dir_down = true;
if (row == key-1)
dir_down = false;

// place the marker
if (rail[row][col] != '*')
result.push_back(rail[row][col++]);

// find the next row using direction flag
dir_down?row++: row--;
}
return result;
}

//driver program to check the above functions
int main()
{
  string code;
  cin >> code;
  int key;
```

```
  cin >> key;
  string cipher = encryptRailFence(code,key);
  cout << "Cipher is: " << cipher << endl;
  cout << "Decrypted message is: " << decryptRailFence(cipher,key);
return 0;
}
```

Output Screenshots:





Result:

Thus, all 3 encryption algorithms have been executed and verified successfully.

CRYPTOGRAPHY AND NETWORK SECURITY
LAB - 2

Name of the Student: SreeDananjay S
Registration Number: 21BAI1807
Slot: L31+L32
Course Code: BCSE309P
Programme: Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning
School: School of Computer Science and Engineering(SCOPE)

Q2)

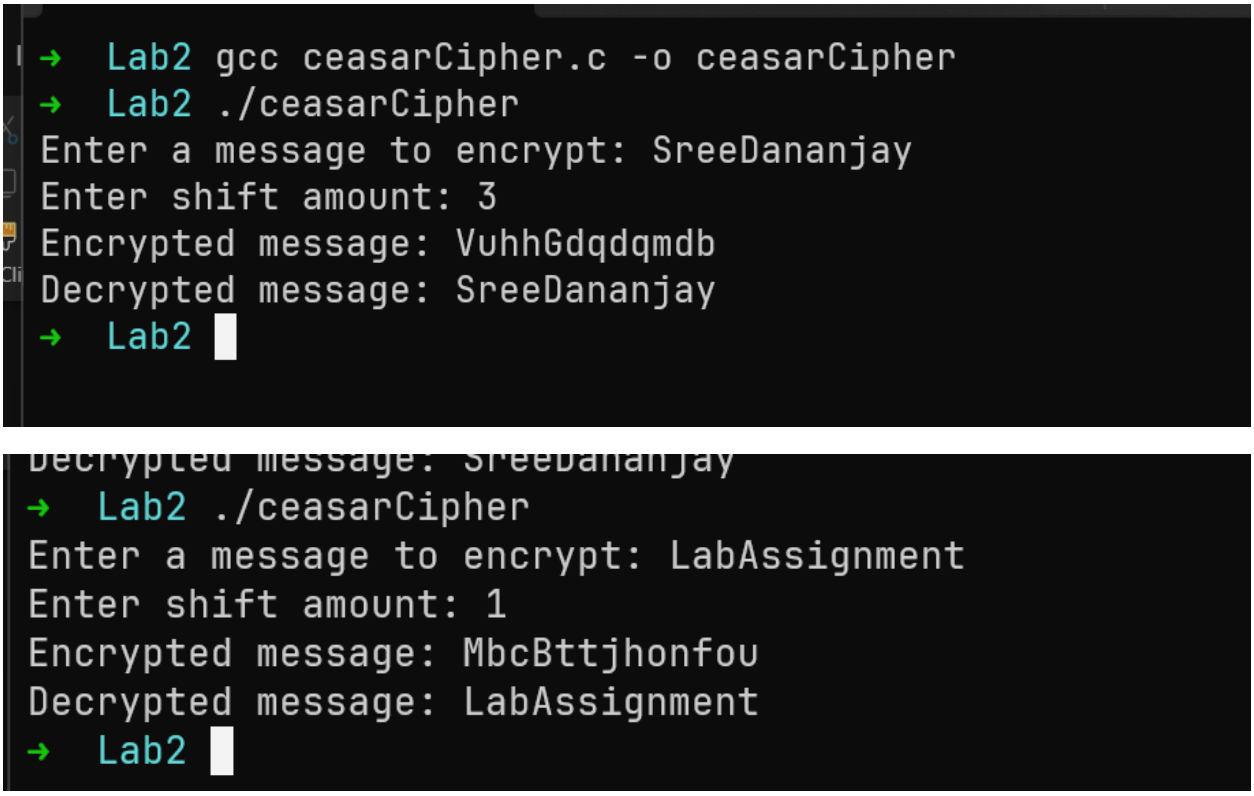
a) Caesar Cipher algorithm implementation in C

Code:

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

void encrypt(char *text, int shift) {
  char ch;
  for (int i = 0; text[i] != '\0'; ++i) {
    ch = text[i];

    // Encrypt uppercase letters
    if (isupper(ch)) {
      text[i] = (ch + shift - 'A') % 26 + 'A';
    }
    // Encrypt lowercase letters
    else if (islower(ch)) {
      text[i] = (ch + shift - 'a') % 26 + 'a';
    }
  }
}

void decrypt(char *text, int shift) {
  encrypt(text, 26 - shift);
}

int main() {
  char text[100];
  int shift;

  printf("Enter a message to encrypt: ");
  fgets(text, sizeof(text), stdin);
  text[strcspn(text, "\n")] = 0; // Remove newline character
```

```
    printf("Enter shift amount: ");
    scanf("%d", &shift);

    encrypt(text, shift);
    printf("Encrypted message: %s\n", text);

    decrypt(text, shift);
    printf("Decrypted message: %s\n", text);

    return 0;
}
```

Sample output Screenshot:



b) Playfair Cipher Implementation in C

Code:

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

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

#define SIZE 5

void generateKeyTable(const char* key, char keyTable[SIZE][SIZE]) {
  int letterExists[26] = {0}; // Track letters already added to the table
  int index = 0;
  int row = 0, col = 0;

  // Add key letters to the table
  for (int i = 0; key[i] != '\0'; i++) {
    char ch = toupper(key[i]);

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

    if (!letterExists[ch - 'A']) {
      keyTable[row][col] = ch;
      letterExists[ch - 'A'] = 1;

      col++;
      if (col == SIZE) {
        col = 0;
        row++;
      }
    }
  }

  // Add remaining letters to the table
  for (char ch = 'A'; ch <= 'Z'; ch++) {
    if (ch == 'J') continue; // Skip 'J'

    if (!letterExists[ch - 'A']) {
      keyTable[row][col] = ch;
      letterExists[ch - 'A'] = 1;

      col++;
      if (col == SIZE) {
```

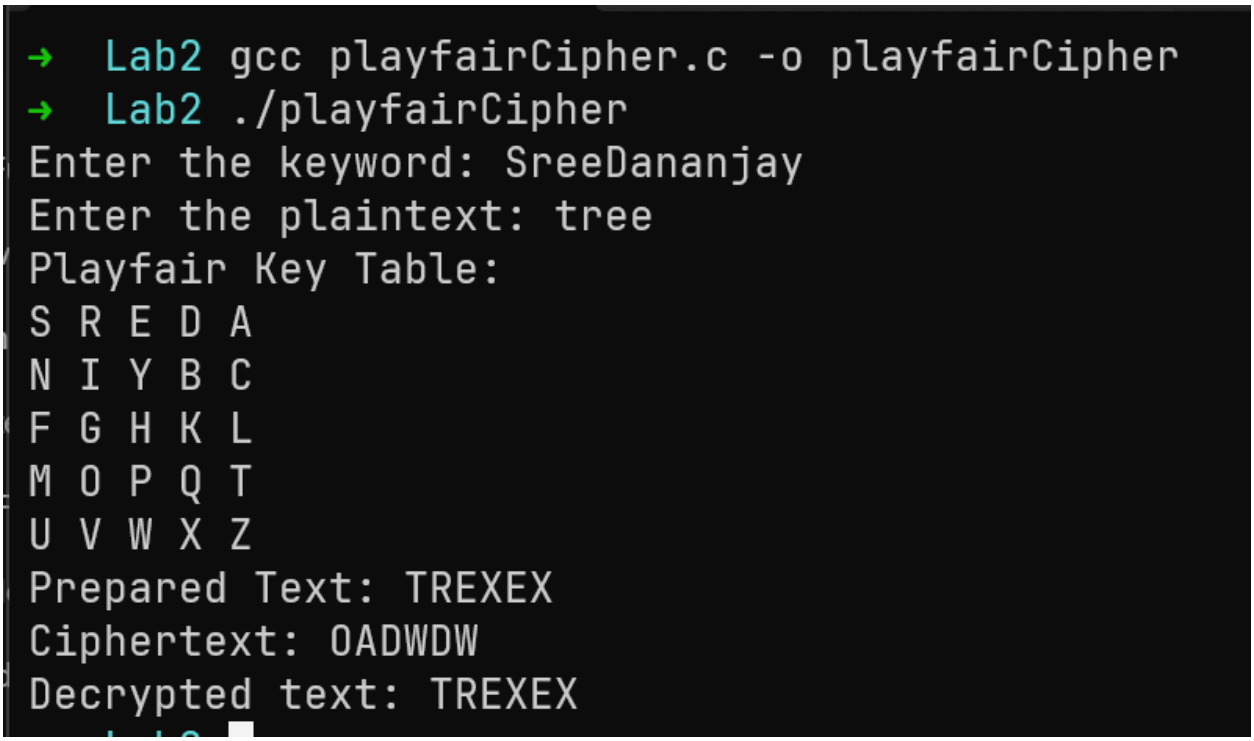
```
            col = 0;
            row++;
        }
    }
  }
}

void printKeyTable(char keyTable[SIZE][SIZE]) {
  printf("Playfair Key Table:\n");
  for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
      printf("%c ", keyTable[i][j]);
    }
    printf("\n");
  }
}

void findPosition(char keyTable[SIZE][SIZE], char ch, int* row, int* col) {
  if (ch == 'J') ch = 'I'; // Treat 'J' as 'I'

  for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
      if (keyTable[i][j] == ch) {
        *row = i;
        *col = j;
        return;
      }
    }
  }
}

void prepareText(const char* input, char* prepared) {
  int len = strlen(input);
  int index = 0;

  for (int i = 0; i < len; i++) {
    char ch = toupper(input[i]);
```

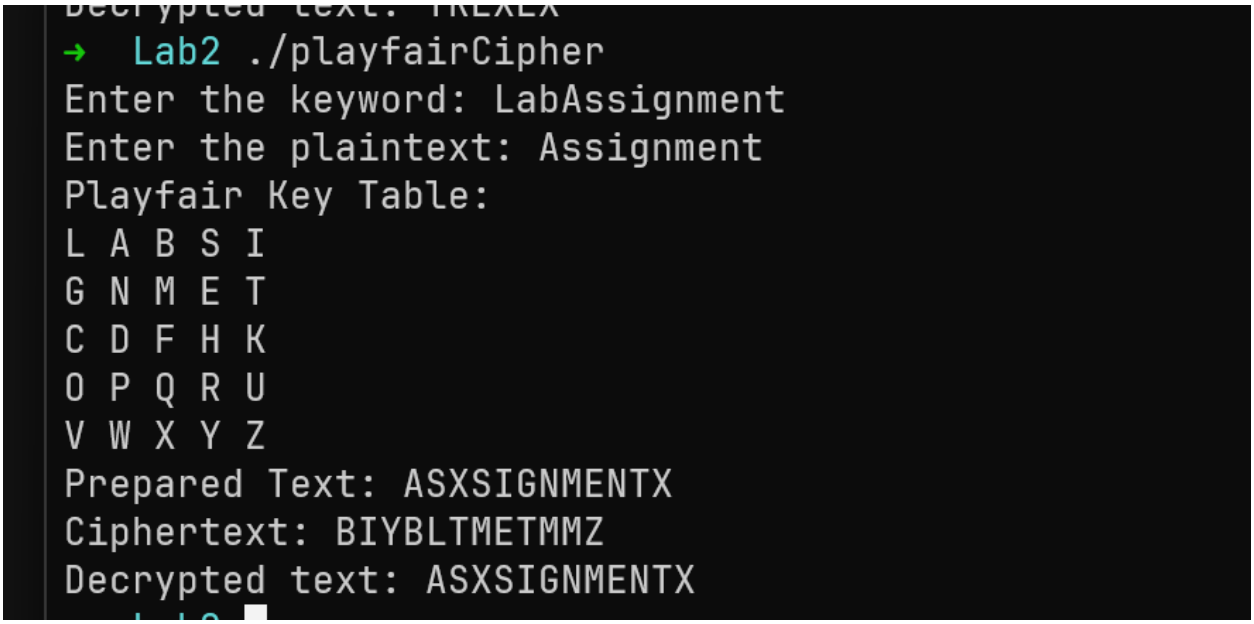
```
        if (ch == 'J') ch = 'I'; // Treat 'J' as 'I'

        if (isalpha(ch)) {
            if (index > 0 && prepared[index - 1] == ch) {
                prepared[index++] = 'X'; // Insert 'X' between duplicate letters
            }
            prepared[index++] = ch;
        }
    }

    if (index % 2 != 0) {
        prepared[index++] = 'X'; // Add 'X' if length is odd
    }

    prepared[index] = '\0';
}

void encryptDigraph(const char digraph[2], char* result, char keyTable[SIZE][SIZE]) {
    int row1, col1, row2, col2;
    findPosition(keyTable, digraph[0], &row1, &col1);
    findPosition(keyTable, digraph[1], &row2, &col2);

    if (row1 == row2) {
        // Same row, shift right
        result[0] = keyTable[row1][(col1 + 1) % SIZE];
        result[1] = keyTable[row2][(col2 + 1) % SIZE];
    } else if (col1 == col2) {
        // Same column, shift down
        result[0] = keyTable[(row1 + 1) % SIZE][col1];
        result[1] = keyTable[(row2 + 1) % SIZE][col2];
    } else {
        // Rectangle case
        result[0] = keyTable[row1][col2];
        result[1] = keyTable[row2][col1];
    }
}

void decryptDigraph(const char digraph[2], char* result, char keyTable[SIZE][SIZE]) {
```

```
    int row1, col1, row2, col2;
    findPosition(keyTable, digraph[0], &row1, &col1);
    findPosition(keyTable, digraph[1], &row2, &col2);

    if (row1 == row2) {
        // Same row, shift left
        result[0] = keyTable[row1][(col1 + SIZE - 1) % SIZE];
        result[1] = keyTable[row2][(col2 + SIZE - 1) % SIZE];
    } else if (col1 == col2) {
        // Same column, shift up
        result[0] = keyTable[(row1 + SIZE - 1) % SIZE][col1];
        result[1] = keyTable[(row2 + SIZE - 1) % SIZE][col2];
    } else {
        // Rectangle case
        result[0] = keyTable[row1][col2];
        result[1] = keyTable[row2][col1];
    }
}

void playfairEncrypt(const char* plaintext, char* ciphertext, char
keyTable[SIZE][SIZE]) {
    char prepared[200];
    prepareText(plaintext, prepared);
    printf("Prepared Text: %s\n", prepared);

    int len = strlen(prepared);
    for (int i = 0; i < len; i += 2) {
        encryptDigraph(&prepared[i], &ciphertext[i], keyTable);
    }
    ciphertext[len] = '\0';
}

void playfairDecrypt(const char* ciphertext, char* plaintext, char
keyTable[SIZE][SIZE]) {
    int len = strlen(ciphertext);
    for (int i = 0; i < len; i += 2) {
        decryptDigraph(&ciphertext[i], &plaintext[i], keyTable);
    }
```

```
    plaintext[len] = '\0';
}

int main() {
    char key[100], plaintext[200], ciphertext[200], decrypted[200];
    char keyTable[SIZE][SIZE];

    printf("Enter the keyword: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = 0; // Remove newline character

    printf("Enter the plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = 0; // Remove newline character

    generateKeyTable(key, keyTable);
    printKeyTable(keyTable);

    playfairEncrypt(plaintext, ciphertext, keyTable);
    printf("Ciphertext: %s\n", ciphertext);

    playfairDecrypt(ciphertext, decrypted, keyTable);
    printf("Decrypted text: %s\n", decrypted);

    return 0;
}
```

Output Screenshots:





Result:

Thus, both encryption algorithms have been executed and verified successfully.