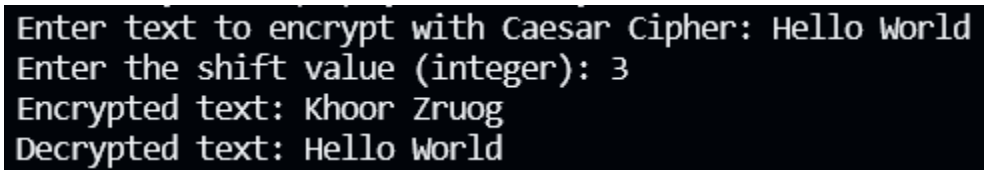
Assignment 1

Caesar Cipher:

Code:-

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

using namespace std;

string caesar_cipher(string text, int shift) {
    for (char &c : text) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            c = (c - base + shift) % 26 + base;
        }
    }
    return text;
}

string caesar_cipher_decrypt(string text, int shift) {
    return caesar_cipher(text, 26 - shift);
}

int main() {
    string text;
    int shift;

    cout << "Enter text to encrypt with Caesar Cipher: ";
    getline(cin, text);
    cout << "Enter the shift value (integer): ";
    cin >> shift;

    string encrypted_text = caesar_cipher(text, shift);
    cout << "Encrypted text: " << encrypted_text << endl;

    string decrypted_text = caesar_cipher_decrypt(encrypted_text, shift);
    cout << "Decrypted text: " << decrypted_text << endl;

    return 0;
}
```

Output:-



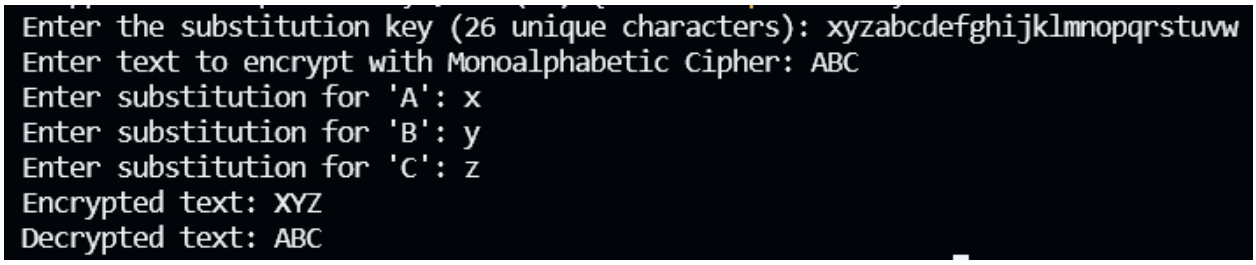
Monoalphabetic Cipher:

Code:-

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

using namespace std;

string monoalphabetic_cipher(string text, unordered_map<char, char> &key) {
    for (char &c : text) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            c = isupper(c) ? toupper(key[tolower(c)]) : key[c];
        }
    }
    return text;
}

string monoalphabetic_cipher_decrypt(string text, unordered_map<char, char> &key) {
    unordered_map<char, char> decryption_key;
    for (auto &pair : key) {
        decryption_key[pair.second] = pair.first;
    }
    return monoalphabetic_cipher(text, decryption_key);
}

int main() {
    string text;
    unordered_map<char, char> key;

    cout << "Enter the substitution key (26 unique characters): ";
    cin >> text;
    cout << "Enter text to encrypt with Monoalphabetic Cipher: ";
    cin.ignore();
    getline(cin, text);

    for (char c : text) {
        if (isalpha(c)) {
```

```
            char base = isupper(c) ? 'A' : 'a';
            char substitution;
            cout << "Enter substitution for '" << c << "': ";
            cin >> substitution;
            key[tolower(c)] = substitution;
        }
    }

    string encrypted_text = monoalphabetic_cipher(text, key);
    cout << "Encrypted text: " << encrypted_text << endl;

    string decrypted_text = monoalphabetic_cipher_decrypt(encrypted_text, key);
    cout << "Decrypted text: " << decrypted_text << endl;

    return 0;
}
```

Output:-



Polyalphabetic (Vigenere) Cipher:

Code:-

```
#include <iostream>

#include <string>

using namespace std;

string vigenere_cipher(string text, string key)

{

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

    {
```

```
        if (isalpha(text[i]))
        {
            char base = isupper(text[i]) ? 'A' : 'a';
            char key_char = key[i % key.length()];
            int shift;
            if(base == 'A')
                shift = toupper(key_char) - base;
            else
                shift = tolower(key_char) - base;
            text[i] = (text[i] - base + shift) % 26 + base;
        }
    }
    return text;
}
string vigenere_cipher_decrypt(string text, string key)
{
    for (int i = 0; i < text.length(); i++)
    {
        if (isalpha(text[i]))
        {
            char base = isupper(text[i]) ? 'A' : 'a';
            char key_char = key[i % key.length()];
            int shift;
            if(base == 'A')
```

```
            shift = toupper(key_char) - base;
        else
            shift = tolower(key_char) - base;
        text[i] = (text[i] - base - shift + 26) % 26 + base;
      }
   }
   return text;
}
int main()
{
   string text, key;
   cout << "Enter the Vigenère key: ";
   cin >> key;
   cout << "Enter text to encrypt with Vigenère Cipher: ";
   cin.ignore();
   getline(cin, text);
   string encrypted_text = vigenere_cipher(text, key);
   cout << "Encrypted text: " << encrypted_text << endl;
   string decrypted_text = vigenere_cipher_decrypt(encrypted_text, key);
   cout << "Decrypted text: " << decrypted_text << endl;
   return 0;
}
```

Output:-

```
Enter the Vigenère key: KEY
Enter text to encrypt with Vigenère Cipher: Hello World
Encrypted text: Rijvs Gspvh
Decrypted text: Hello World
```

Vernam Cipher:

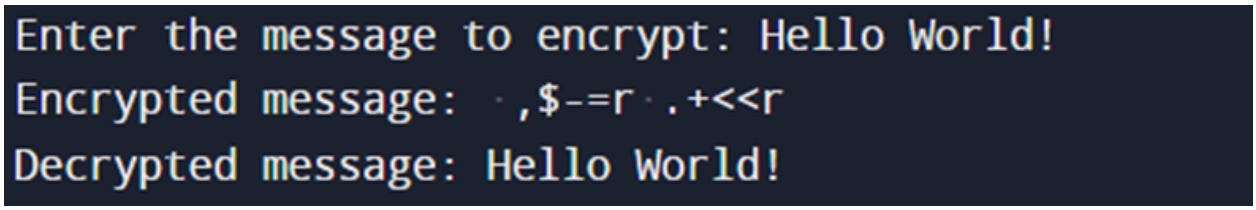
Code:-

```
#include <iostream>

#include <string>

#include <cstdlib>

#include <ctime>

using namespace std;


// Function to generate a random key of the same length as the message

string generateRandomKey(int length) {

    string key;

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

        key += (char)(rand() % 26 + 'A'); // Assuming uppercase letters

    }

    return key;

}


// Function to perform Vernam cipher encryption
```

```
string vernamEncrypt(const string& message, const string& key) {
    string ciphertext;
    for (size_t i = 0; i < message.length(); ++i) {
        char encryptedChar = message[i] ^ key[i];
        ciphertext += encryptedChar;
    }
    return ciphertext;
}

// Function to perform Vernam cipher decryption
string vernamDecrypt(const string& ciphertext, const string& key) {
    string decryptedText;
    for (size_t i = 0; i < ciphertext.length(); ++i) {
        char decryptedChar = ciphertext[i] ^ key[i];
        decryptedText += decryptedChar;
    }
    return decryptedText;
}

int main() {
    // Seed the random number generator
    srand(static_cast<unsigned>(time(0)));

    string message;
```

```
    cout << "Enter the message to encrypt: ";
    getline(cin, message);

    // Generate a random key of the same length as the message
    string key = generateRandomKey(message.length());

    // Encrypt the message
    string ciphertext = vernamEncrypt(message, key);
    cout << "Encrypted message: " << ciphertext << endl;

    // Decrypt the message using the same key
    string decryptedText = vernamDecrypt(ciphertext, key);
    cout << "Decrypted message: " << decryptedText << endl;

    return 0;
}
```

Output:-



Rail Fence Cipher:

Code:-

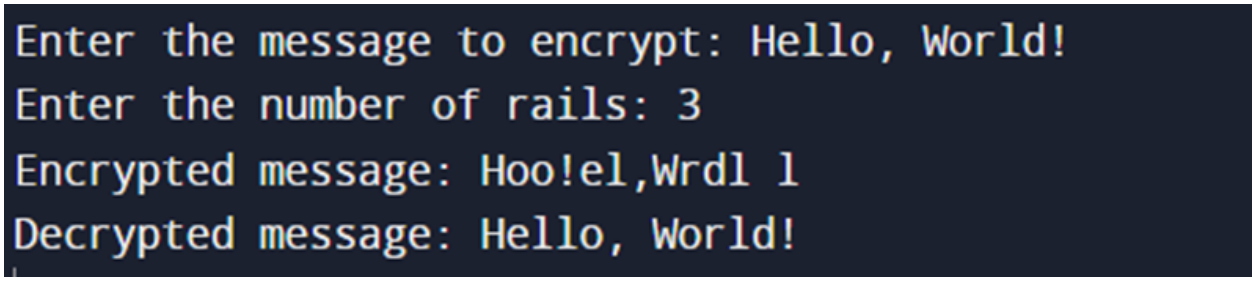
```
#include <iostream>

#include <string>

#include <vector>

using namespace std;


// Function to perform Rail Fence cipher encryption

string railFenceEncrypt(const string& message, int railCount) {

    vector<string> rails(railCount, "");

    int currentRail = 0;

    bool downDirection = true;

    for (char c : message) {

        rails[currentRail] += c;

        if (currentRail == 0)

            downDirection = true;

        else if (currentRail == railCount - 1)

            downDirection = false;

        if (downDirection)

            currentRail++;

        else

            currentRail--;

    }
```

```
    string ciphertext;
    for (const string& rail : rails) {
        ciphertext += rail;
    }
    return ciphertext;
}

// Function to perform Rail Fence cipher decryption
string railFenceDecrypt(const string& ciphertext, int railCount) {
    vector<string> rails(railCount, "");
    int currentRail = 0;
    bool downDirection = true;
    // Calculate the length of each rail
    vector<int> railLengths(railCount, 0);
    int railIndex = 0;
    for (char c : ciphertext) {
        railLengths[railIndex]++;
        if (railIndex == 0)
            downDirection = true;
        else if (railIndex == railCount - 1)
            downDirection = false;
        if (downDirection)
            railIndex++;
        else
```

```
            railIndex--;
        }
        int currentIndex = 0;
        for (int railIndex = 0; railIndex < railCount; ++railIndex) {
            rails[railIndex] = ciphertext.substr(currentIndex, railLengths[railIndex]);
            currentIndex += railLengths[railIndex];
        }
        string plaintext;
        currentRail = 0;
        downDirection = true;
        for (int i = 0; i < ciphertext.length(); ++i) {
            plaintext += rails[currentRail][0];
            rails[currentRail].erase(0, 1);
            if (currentRail == 0)
                downDirection = true;
            else if (currentRail == railCount - 1)
                downDirection = false;
            if (downDirection)
                currentRail++;
            else
                currentRail--;
        }
        return plaintext;
    }
```

```
int main() {

    string message;

    int railCount;

    cout << "Enter the message to encrypt: ";

    getline(cin, message);

    cout << "Enter the number of rails: ";

    cin >> railCount;

    string ciphertext = railFenceEncrypt(message, railCount);

    cout << "Encrypted message: " << ciphertext << endl;

    string decryptedText = railFenceDecrypt(ciphertext, railCount);

    cout << "Decrypted message: " << decryptedText << endl;

    return 0;

}
```

Output:-



Conclusion:-

In conclusion, the classical encryption methods presented here offer a variety of techniques to secure data and information. Each of these methods has its strengths and weaknesses, making them suitable for different use cases:

- Caesar Cipher: The Caesar Cipher is a straightforward substitution cipher. It's easy to implement and understand but not very secure. It's suitable for educational purposes or cases where minimal security is required.
- Monoalphabetic Cipher (Substitution Cipher): Monoalphabetic substitution ciphers are stronger than Caesar Ciphers as they involve a one-to-one mapping of characters. However, they are still vulnerable to frequency analysis attacks and unsuitable for securing sensitive data.
- Polyalphabetic Cipher (Vigenère Cipher): The Vigenère Cipher improves security compared to the previous ciphers by using a keyword to perform multiple shifts. However, it can still be vulnerable to crucial length discovery and frequency analysis attacks.
- Vernam Cipher (One-Time Pad): The Vernam Cipher, also known as the one-time pad, is theoretically unbreakable when used with truly random keys as long as the message. However, it is impractical for most real-world scenarios due to the requirement for perfectly random keys.
- Rail Fence Cipher: The Rail Fence Cipher is a transposition cipher that rearranges the characters in a zigzag pattern. It provides essential security but is vulnerable to brute-force attacks and might not be suitable for securing sensitive data.