PRACTICAL NO 01

PART 1

```
#include <iostream>
#include <string>
using namespace std;

// Caesar Cipher Encryption
string caesarEncrypt(string text, int key) {
    string result = "";
    for (char c : text) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            result += (c - base + key) % 26 +
base;
        } else {
            result += c;
        }
    }
    return result;
}

// Caesar Cipher Decryption
string caesarDecrypt(string text, int key)
{
    return caesarEncrypt(text, 26 - (key %
26));
}

// Modified Caesar Cipher Encryption
string modifiedCaesarEncrypt(string text,
int key) {
    string result = "";
    for (size_t i = 0; i < text.length(); ++i) {
        char c = text[i];
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            int shift = (key + i % 5) % 26;
            result += (c - base + shift) % 26 +
base;
        } else {
            result += c;
        }
    }
    return result;
}

// Modified Caesar Cipher Decryption
string modifiedCaesarDecrypt(string text,
int key) {
    string result = "";
    for (size_t i = 0; i < text.length(); ++i) {
        char c = text[i];
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            int shift = (key + i % 5) % 26;
            result += (c - base - shift + 26) %
26 + base;
        } else {
            result += c;
        }
    }
    return result;
}
```

```
// Vigenère Cipher Encryption
string vigenereEncrypt(string text, string key) {
    string result = "";
    int keyLen = key.length();
    int j = 0;
    for (char c : text) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            char keyChar = toupper(key[j % keyLen]) - 'A';
            result += (c - base + keyChar) % 26 + base;
            j++;
        } else {
            result += c;
        }
    }
    return result;
}

// Vigenère Cipher Decryption
string vigenereDecrypt(string text, string key) {
    string result = "";
    int keyLen = key.length();
    int j = 0;
    for (char c : text) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            char keyChar = toupper(key[j % keyLen]) - 'A';
            result += (c - base - keyChar + 26) % 26 + base;
            j++;
        } else {
            result += c;
        }
    }
    return result;
}

int main() {
    int choice;
    string plaintext, keyStr;
    int keyInt;

    cout << "===== CIPHER MENU =====\n";
    cout << "1. Caesar Cipher\n";
    cout << "2. Modified Caesar Cipher\n";
    cout << "3. Vigenère Cipher\n";
    cout << "Enter your choice (1-3): ";
    cin >> choice;
    cin.ignore(); // Clear input buffer

    switch (choice) {
        case 1:
            cout << "Enter plaintext: ";
            getline(cin, plaintext);
            cout << "Enter key (int): ";
            cin >> keyInt;
            {
                string encrypted = caesarEncrypt(plaintext, keyInt);
```

```
            string decrypted =
caesarDecrypt(encrypted, keyInt);
            cout << "Encrypted text: " <<
encrypted << endl;
            cout << "Decrypted text: " <<
decrypted << endl;
        }
        break;

    case 2:
        cout << "Enter plaintext: ";
        getline(cin, plaintext);
        cout << "Enter base key (int): ";
        cin >> keyInt;
        {
            string encrypted =
modifiedCaesarEncrypt(plaintext, keyInt);
            string decrypted =
modifiedCaesarDecrypt(encrypted,
keyInt);
            cout << "Encrypted text: " <<
encrypted << endl;
            cout << "Decrypted text: " <<
decrypted << endl;
        }
        break;

    case 3:
        cout << "Enter plaintext: ";
        getline(cin, plaintext);
        cout << "Enter keyword: ";
        getline(cin, keyStr);
        {
            string encrypted =
vigenereEncrypt(plaintext, keyStr);
            string decrypted =
vigenereDecrypt(encrypted, keyStr);
            cout << "Encrypted text: " <<
encrypted << endl;
            cout << "Decrypted text: " <<
decrypted << endl;
        }
        break;

    default:
        cout << "Invalid choice!" << endl;
  }

  return 0;
}
```

```
PS E:\Practicals\Practicals 7th Sem\CNS\Practical 1> cd "e:\Practicals\Practicals 7th Sem\CNS\Practical 1\" ; if ($?) { g++ practical1.cpp -o practical1
if ($?) { .\practical1 }
===== CIPHER MENU =====
1. Caesar Cipher
2. Modified Caesar Cipher
3. Vigen├¿re Cipher
Enter your choice (1-3): 1
Enter plaintext: helloworld
Enter key (int): 3
Encrypted text: khoorzruog
Decrypted text: helloworld
```

```
PS E:\Practicals\Practicals 7th Sem\CNS\Practical 1> cd "e:\Practicals\Practicals 7th Sem\CNS\Practical 1\" ; if ($?) { g++ practical1.cpp -o practical1 }
if ($?) { .\practical1 }
===== CIPHER MENU =====
1. Caesar Cipher
2. Modified Caesar Cipher
3. Vigen├¿re Cipher
Enter your choice (1-3): 2
Enter plaintext: music
Enter base key (int): 1,2
Encrypted text: nwvmh
Decrypted text: music
PS E:\Practicals\Practicals 7th Sem\CNS\Practical 1>
```

```
PS E:\Practicals\Practicals 7th Sem\CNS\Practical 1> cd "e:\Practicals\Practicals 7th Sem\CNS\Practical 1\" ; if ($?) { g++ practical1.cpp -o practical1
if ($?) { .\practical1 }
===== CIPHER MENU =====
1. Caesar Cipher
2. Modified Caesar Cipher
3. Vigen├¿re Cipher
Enter your choice (1-3): 3
Enter plaintext: apple
Enter keyword: snow
Encrypted text: scdhw
Decrypted text: apple
```

PLAYFIRE CIPHER

```
//PLAYFIRE CIPHER


#include <iostream>
#include <vector>
#include <algorithm>
#include <cctype>
using namespace std;
char matrix[5][5];
// Check if character already exists in the key matrix
bool exists(char c, string& used) {
    return used.find(c) != string::npos;
}
// Create 5x5 Playfair matrix from key
void generateMatrix(string key) {
    string used = "";
    key.erase(remove(key.begin(), key.end(), 'J'), key.end()); // Replace J with I rule
    key += "ABCDEFGHIKLMNOPQRSTUVWXYZ";
    int idx = 0;
    for (char c : key) {
        c = toupper(c);
        if (isalpha(c) && !exists(c, used)) {
            used += c;
            matrix[idx / 5][idx % 5] = c;
            idx++;
        }
    }
}// Find row and column of a character in matrix
void findPosition(char c, int &row, int &col) {
    if (c == 'J') c = 'I';
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 5; j++)
            if (matrix[i][j] == c) {
                row = i;
                col = j;
                return;
            }
}
// Prepare text into digraphs, insert bogus letter if needed
vector<pair<char, char>> prepareText(string text, char bogus) {
    vector<pair<char, char>> pairs;
    text.erase(remove(text.begin(), text.end(), ' '), text.end());
    for (char &c : text) {
        if (c == 'J') c = 'I';
        c = toupper(c);
    }
    for (size_t i = 0; i < text.length(); ) {
        char a = text[i];
        char b;
        if (i + 1 < text.length()) {
            b = text[i + 1];
            if (a == b) {
                b = bogus;
                i++;
            } else {
                i += 2;
```

```
            }
        } else {
            b = bogus;
    i++;
        }
        pairs.push_back({a, b});
    }
    return pairs;
}
// Encrypt text using Playfair rules
string encrypt(string text, char bogus) {
    vector<pair<char, char>> pairs =
prepareText(text, bogus);
    string cipher = "";
    for (auto p : pairs) {
        int r1, c1, r2, c2;
        findPosition(p.first, r1, c1);
        findPosition(p.second, r2, c2);
        if (r1 == r2) {
            cipher += matrix[r1][(c1 + 1) % 5];
            cipher += matrix[r2][(c2 + 1) % 5];
        } else if (c1 == c2) {
            cipher += matrix[(r1 + 1) % 5][c1];
            cipher += matrix[(r2 + 1) % 5][c2];
        } else {
            cipher += matrix[r1][c2];
            cipher += matrix[r2][c1];
        }
    }
    return cipher;
}
// Decrypt ciphertext using Playfair rules
string decrypt(string cipher) {
    string plain = "";
    for (size_t i = 0; i < cipher.length(); i +=
2) {
   char a = cipher[i];
        char b = cipher[i + 1];
        int r1, c1, r2, c2;
        findPosition(a, r1, c1);
        findPosition(b, r2, c2);
        if (r1 == r2) {
            plain += matrix[r1][(c1 + 4) % 5];
            plain += matrix[r2][(c2 + 4) % 5];
        } else if (c1 == c2) {
            plain += matrix[(r1 + 4) % 5][c1];
            plain += matrix[(r2 + 4) % 5][c2];
        } else {
            plain += matrix[r1][c2];
            plain += matrix[r2][c1];
        }
    }
    return plain;}
int main() {
    string key, plaintext, ciphertext;
    char bogus;
    cout << "Enter the key: ";
    getline(cin, key);
    cout << "Enter the bogus/filler letter
(e.g. X): ";
    cin >> bogus;
    bogus = toupper(bogus);
    cin.ignore(); // to consume newline after
bogus letter inpuT
    cout << "Enter the plaintext: ";
```

```
    getline(cin, plaintext);
    generateMatrix(key);
    cout << "\nPlayfair Matrix:\n";
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    ciphertext = encrypt(plaintext, bogus);
    cout << "\nEncrypted Ciphertext: " << ciphertext << endl;
    string decrypted = decrypt(ciphertext);
    cout << "Decrypted Text: " << decrypted << endl;
    return 0;
}
```

```
PS E:\Practicals\Practicals 7th Sem\CNS\Practical 1> cd "e:\Practicals\Practicals 7th Sem\CNS\Practical 1\" ; if ($?) { g++ playfire.cpp -o playfire } ;
$?) { .\playfire }
Enter the key: computer
Enter the bogus/filler letter (e.g. X): X
Enter the plaintext: tables

Playfair Matrix:
C O M P U
T E R A B
D F G H I
K L N Q S
V W X Y Z

Encrypted Ciphertext: EBESBL
Decrypted Text: TABLES
```