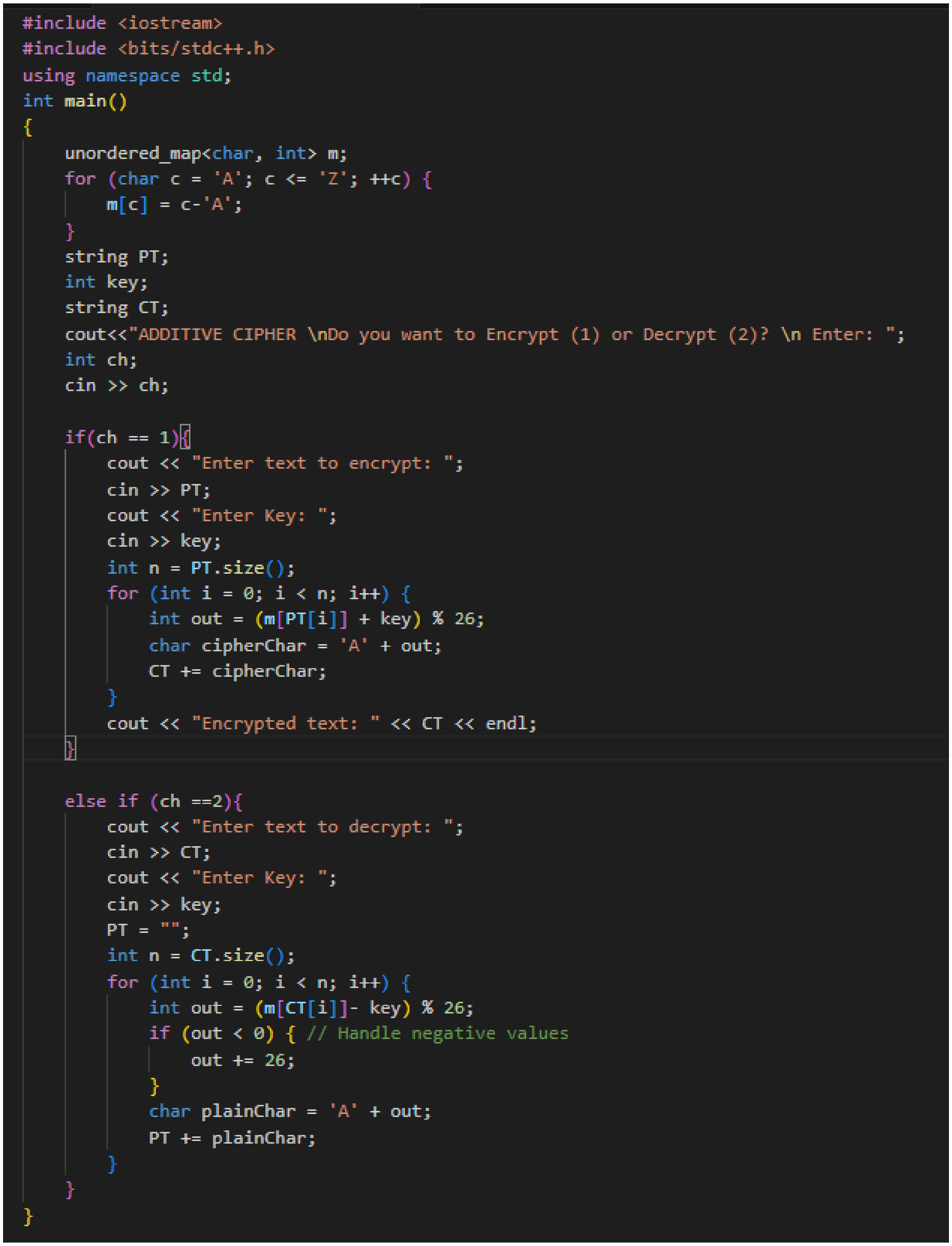
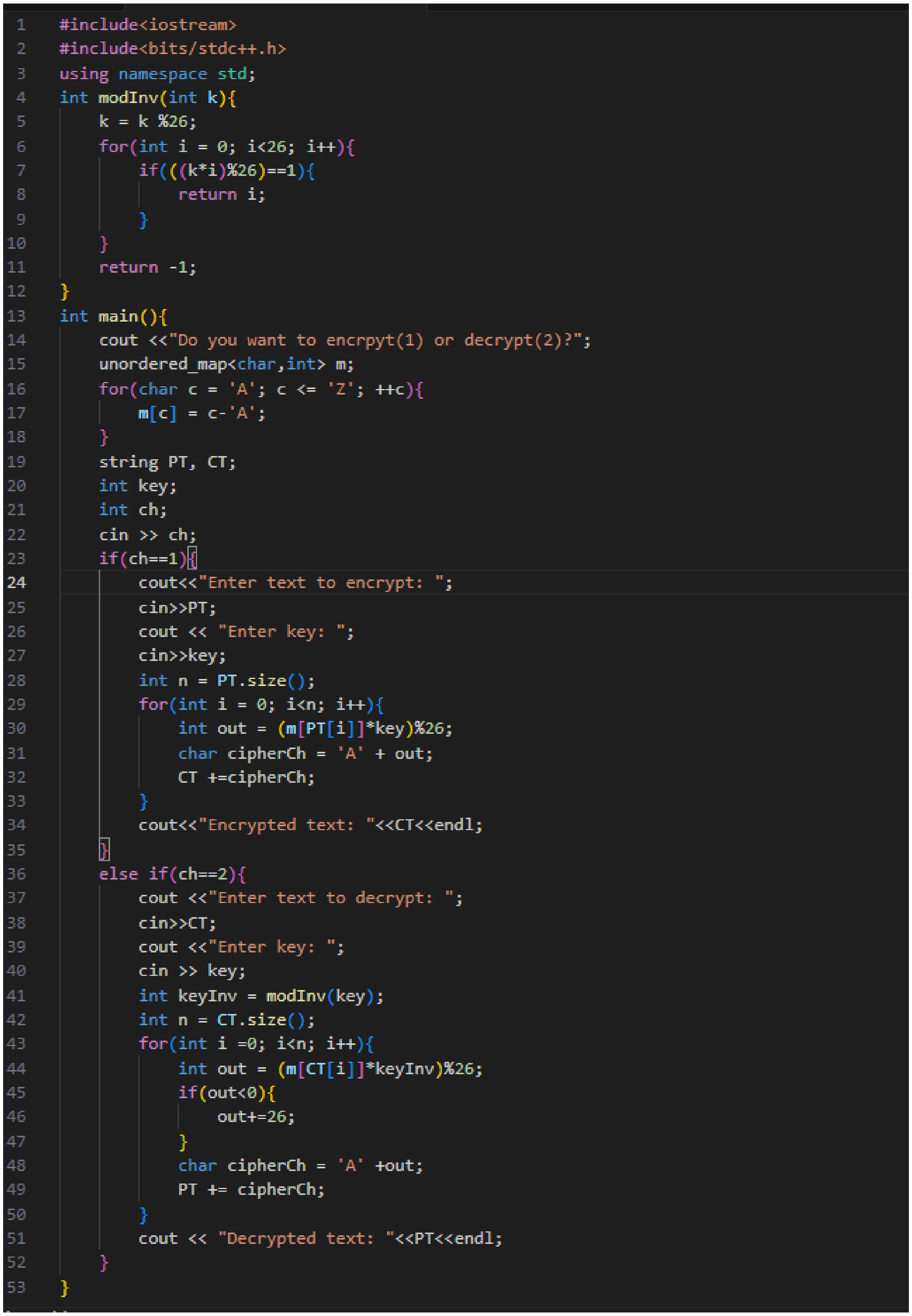
Codes

Additive Ciphers



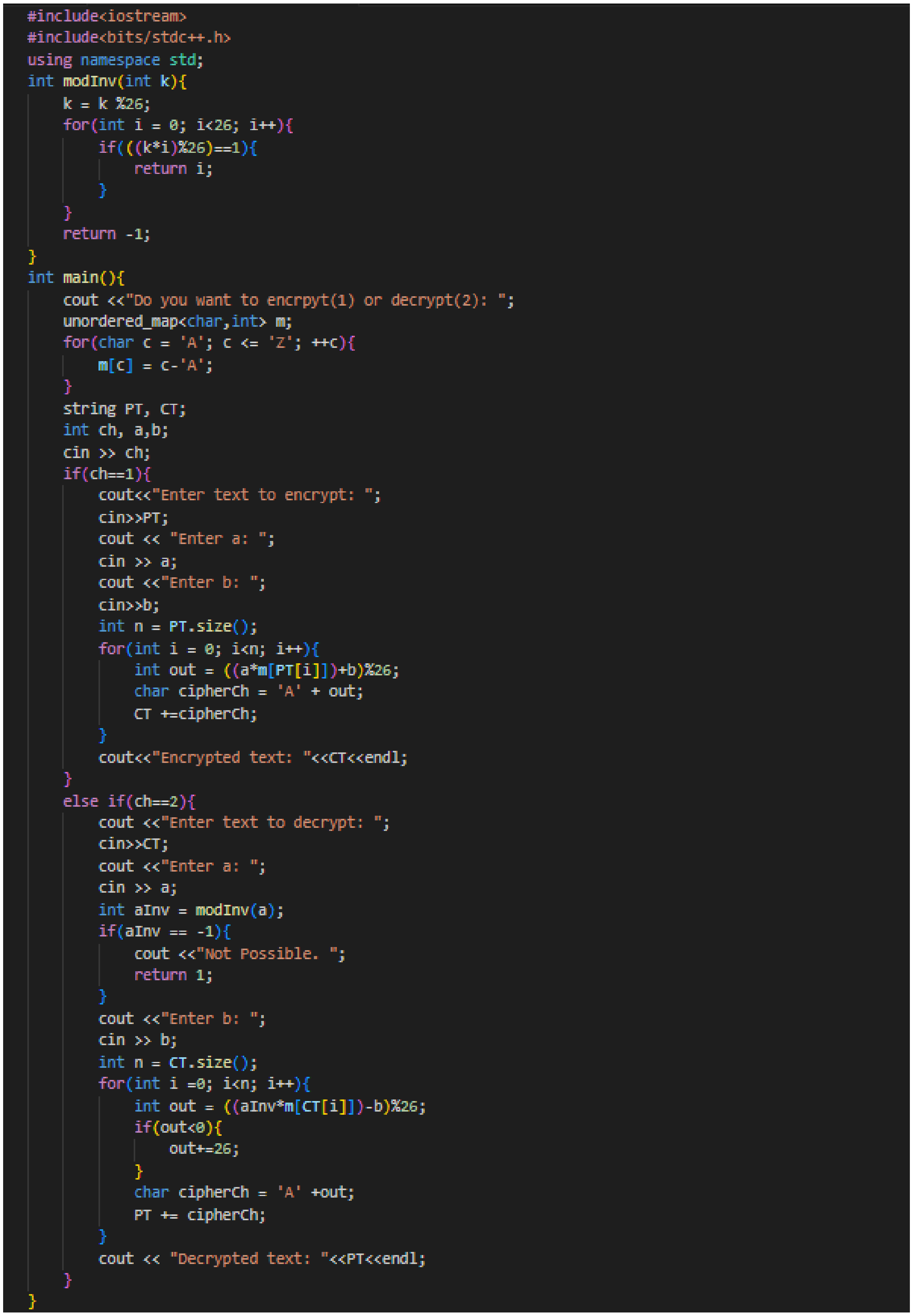
```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
int main()
{
    unordered_map<char, int> m;
    for (char c = 'A'; c <= 'Z'; ++c) {
        m[c] = c-'A';
    }
    string PT;
    int key;
    string CT;
    cout<<"ADDITIVE CIPHER \nDo you want to Encrypt (1) or Decrypt (2)? \n Enter: ";
    int ch;
    cin >> ch;

    if(ch == 1){
        cout << "Enter text to encrypt: ";
        cin >> PT;
        cout << "Enter Key: ";
        cin >> key;
        int n = PT.size();
        for (int i = 0; i < n; i++) {
            int out = (m[PT[i]] + key) % 26;
            char cipherChar = 'A' + out;
            CT += cipherChar;
        }
        cout << "Encrypted text: " << CT << endl;
    }

    else if (ch ==2){
        cout << "Enter text to decrypt: ";
        cin >> CT;
        cout << "Enter Key: ";
        cin >> key;
        PT = "";
        int n = CT.size();
        for (int i = 0; i < n; i++) {
            int out = (m[CT[i]]- key) % 26;
            if (out < 0) { // Handle negative values
                out += 26;
            }
            char plainChar = 'A' + out;
            PT += plainChar;
        }
    }
}
```

Multiplicative Ciphers



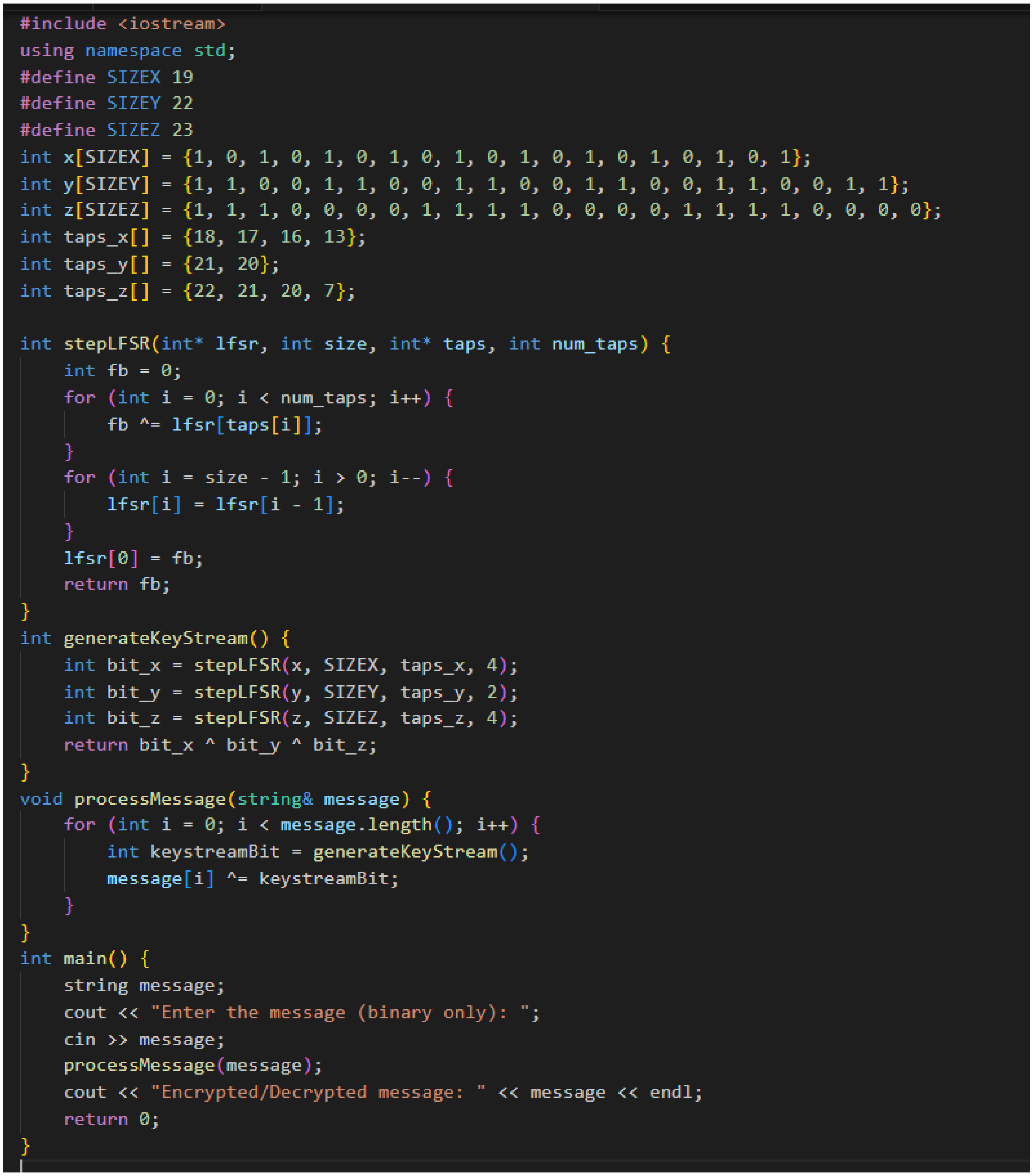
```cpp
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int modInv(int k){
    k = k %26;
    for(int i = 0; i<26; i++){
        if(((k*i)%26)==1){
            return i;
        }
    }
    return -1;
}
int main(){
    cout <<"Do you want to encrpyt(1) or decrypt(2)?";
    unordered_map<char,int> m;
    for(char c = 'A'; c <= 'Z'; ++c){
        m[c] = c-'A';
    }
    string PT, CT;
    int key;
    int ch;
    cin >> ch;
    if(ch==1){
        cout<<"Enter text to encrypt: ";
        cin>>PT;
        cout << "Enter key: ";
        cin>>key;
        int n = PT.size();
        for(int i = 0; i<n; i++){
            int out = (m[PT[i]]*key)%26;
            char cipherCh = 'A' + out;
            CT +=cipherCh;
        }
        cout<<"Encrypted text: "<<CT<<endl;
    }
    else if(ch==2){
        cout <<"Enter text to decrypt: ";
        cin>>CT;
        cout <<"Enter key: ";
        cin >> key;
        int keyInv = modInv(key);
        int n = CT.size();
        for(int i =0; i<n; i++){
            int out = (m[CT[i]]*keyInv)%26;
            if(out<0){
                out+=26;
            }
            char cipherCh = 'A' +out;
            PT += cipherCh;
        }
        cout << "Decrypted text: "<<PT<<endl;
    }
}
```

Affine Ciphers



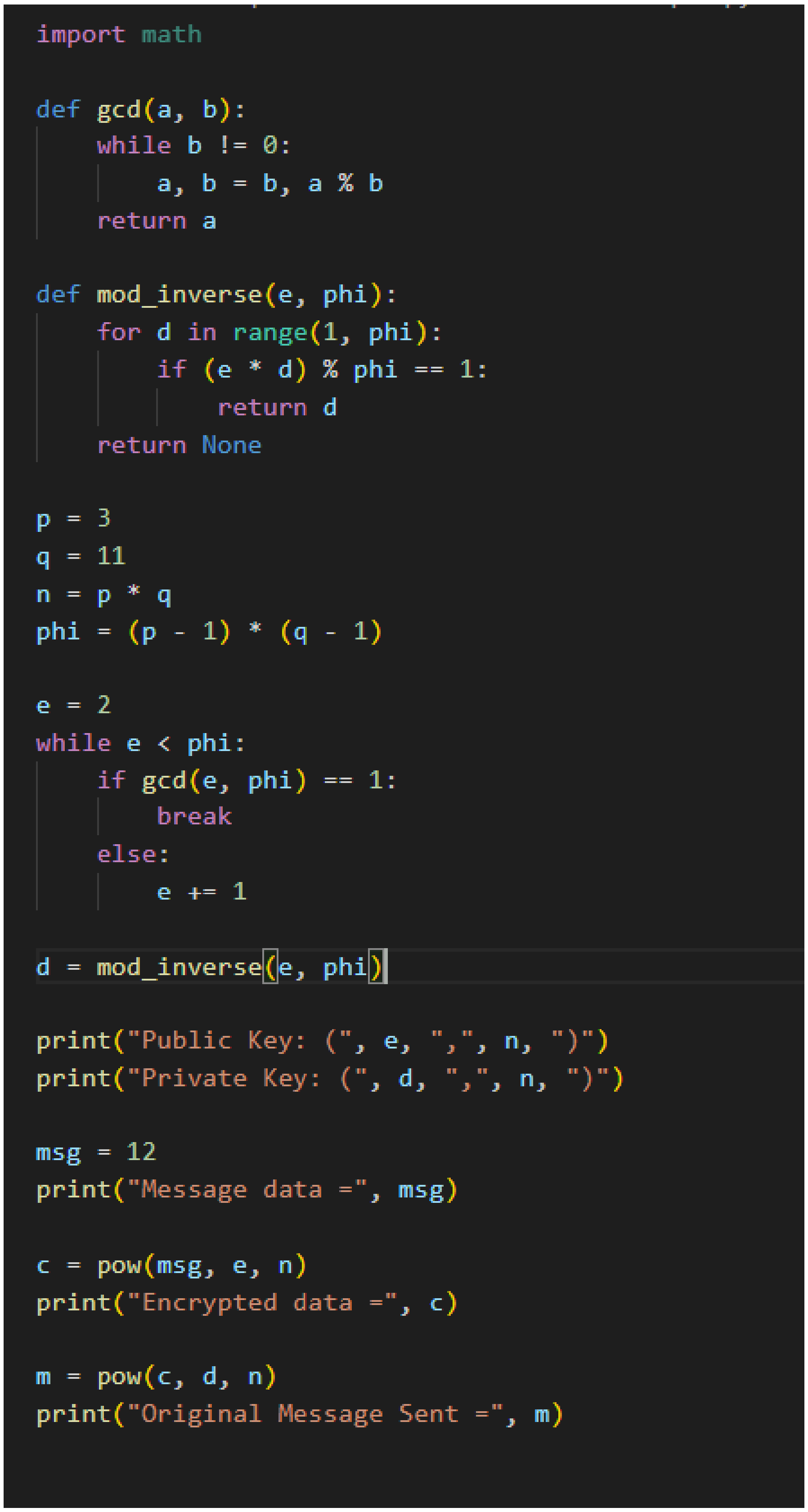
```cpp
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int modInv(int k){
    k = k %26;
    for(int i = 0; i<26; i++){
        if(((k*i)%26)==1){
            return i;
        }
    }
    return -1;
}
int main(){
    cout <<"Do you want to encrpyt(1) or decrypt(2): ";
    unordered_map<char,int> m;
    for(char c = 'A'; c <= 'Z'; ++c){
        m[c] = c-'A';
    }
    string PT, CT;
    int ch, a,b;
    cin >> ch;
    if(ch==1){
        cout<<"Enter text to encrypt: ";
        cin>>PT;
        cout << "Enter a: ";
        cin >> a;
        cout <<"Enter b: ";
        cin>>b;
        int n = PT.size();
        for(int i = 0; i<n; i++){
            int out = ((a*m[PT[i]])+b)%26;
            char cipherCh = 'A' + out;
            CT +=cipherCh;
        }
        cout<<"Encrypted text: "<<CT<<endl;
    }
    else if(ch==2){
        cout <<"Enter text to decrypt: ";
        cin>>CT;
        cout <<"Enter a: ";
        cin >> a;
        int aInv = modInv(a);
        if(aInv == -1){
            cout <<"Not Possible. ";
            return 1;
        }
        cout <<"Enter b: ";
        cin >> b;
        int n = CT.size();
        for(int i =0; i<n; i++){
            int out = ((aInv*m[CT[i]])-b)%26;
            if(out<0){
                out+=26;
            }
            char cipherCh = 'A' +out;
            PT += cipherCh;
        }
        cout << "Decrypted text: "<<PT<<endl;
    }
}
```

A5/1



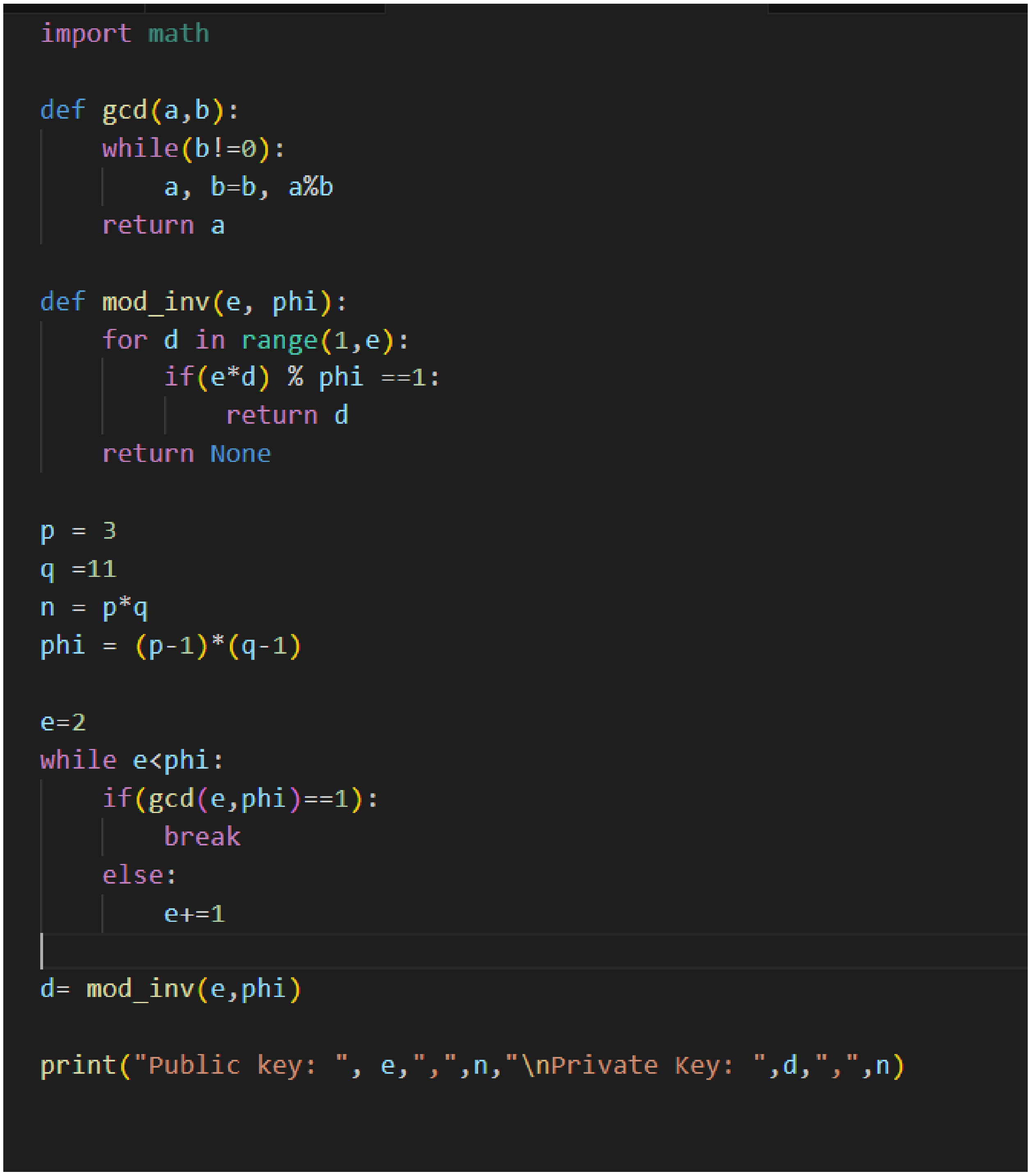
```cpp
#include <iostream>
using namespace std;
#define SIZEX 19
#define SIZEY 22
#define SIZEZ 23
int x[SIZEX] = {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1};
int y[SIZEY] = {1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1};
int z[SIZEZ] = {1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0};
int taps_x[] = {18, 17, 16, 13};
int taps_y[] = {21, 20};
int taps_z[] = {22, 21, 20, 7};

int stepLFSR(int* lfsr, int size, int* taps, int num_taps) {
    int fb = 0;
    for (int i = 0; i < num_taps; i++) {
        fb ^= lfsr[taps[i]];
    }
    for (int i = size - 1; i > 0; i--) {
        lfsr[i] = lfsr[i - 1];
    }
    lfsr[0] = fb;
    return fb;
}
int generateKeyStream() {
    int bit_x = stepLFSR(x, SIZEX, taps_x, 4);
    int bit_y = stepLFSR(y, SIZEY, taps_y, 2);
    int bit_z = stepLFSR(z, SIZEZ, taps_z, 4);
    return bit_x ^ bit_y ^ bit_z;
}
void processMessage(string& message) {
    for (int i = 0; i < message.length(); i++) {
        int keystreamBit = generateKeyStream();
        message[i] ^= keystreamBit;
    }
}
int main() {
    string message;
    cout << "Enter the message (binary only): ";
    cin >> message;
    processMessage(message);
    cout << "Encrypted/Decrypted message: " << message << endl;
    return 0;
}
```

RSA



```
import math

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def mod_inverse(e, phi):
    for d in range(1, phi):
        if (e * d) % phi == 1:
            return d
    return None

p = 3
q = 11
n = p * q
phi = (p - 1) * (q - 1)

e = 2
while e < phi:
    if gcd(e, phi) == 1:
        break
    else:
        e += 1

d = mod_inverse(e, phi)

print("Public Key: (", e, ",", n, ")")
print("Private Key: (", d, ",", n, ")")

msg = 12
print("Message data =", msg)

c = pow(msg, e, n)
print("Encrypted data =", c)

m = pow(c, d, n)
print("Original Message Sent =", m)
```

Diffie Hellman



```
import math

def gcd(a,b):
    while(b!=0):
        a, b=b, a%b
    return a

def mod_inv(e, phi):
    for d in range(1,e):
        if(e*d) % phi ==1:
            return d
    return None

p = 3
q =11
n = p*q
phi = (p-1)*(q-1)

e=2
while e<phi:
    if(gcd(e,phi)==1):
        break
    else:
        e+=1

d= mod_inv(e,phi)

print("Public key: ", e,",",n,"\nPrivate Key: ",d,",",n)
```

Row transposition ciphers and Columnar transposition ciphers

```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
string rowEnc(const string& text, int numRows) {
    int n = text.size();
    int numCols = (n + numRows - 1) / numRows;
    vector<string> grid(numRows, string(numCols, ' '));
    for (int i = 0; i < n; ++i) {
        grid[i % numRows][i / numRows] = text[i];
    }
    string encrypted;
    for (const auto& row : grid) {
        encrypted += row;
    }
    return encrypted;
}

string rowDec(const string& text, int numRows) {
    int n = text.size();
    int numCols = (n + numRows - 1) / numRows;
    vector<string> grid(numRows, string(numCols, ' '));
    int idx = 0;
    for (int col = 0; col < numCols; ++col) {
        for (int row = 0; row < numRows; ++row) {
            if (idx < n) {
                grid[row][col] = text[idx++];
            }
        }
    }
    string decrypted;
    for (const auto& row : grid) {
        decrypted += row;
    }
    return decrypted;
}

string colEn(const string& text, const vector<int>& key) {
    int n = text.size();
    int numCols = key.size();
    int numRows = (n + numCols - 1) / numCols;
    vector<string> grid(numRows, string(numCols, ' '));
    for (int i = 0; i < n; ++i) {
        grid[i / numCols][i % numCols] = text[i];
    }
    string encrypted;
    for (int col : key) {
        for (int row = 0; row < numRows; ++row) {
            encrypted += grid[row][col];
        }
    }
    return encrypted;
}

string colDec(const string& text, const vector<int>& key) {
    int n = text.size();
    int numCols = key.size();
    int numRows = (n + numCols - 1) / numCols;
    vector<string> grid(numRows, string(numCols, ' '));
    int idx = 0;
    for (int col : key) {
        for (int row = 0; row < numRows; ++row) {
            if (idx < n) {
                grid[row][col] = text[idx++];
            }
        }
    }
    string decrypted;
    for (const auto& row : grid) {
        decrypted += row;
    }
    return decrypted;
}
int main() {
    cout << "Choose cipher type: Row(1) or Column(2): ";
    int choice;
    cin >> choice;
    if (choice == 1) {
        int numRows;
        cout << "Enter the number of rows for Row Transposition Cipher: ";
        cin >> numRows;
        cout << "Encrypt(1) or Decrypt(2): ";
        int action;
        cin >> action;
        string text;
        cout << "Enter text (no spaces): ";
        cin >> text;
        if (action == 1) {
            string encryptedText = rowEnc(text, numRows);
            cout << "Encrypted Text: " << encryptedText << endl;
        } else if (action == 2) {
            string decryptedText = rowDec(text, numRows);
            cout << "Decrypted Text: " << decryptedText << endl;
        }
    } else if (choice == 2) {
        int keyLength;
        cout << "Enter the length of the column key: ";
        cin >> keyLength;
        vector<int> key(keyLength);
        cout << "Enter the column key (0-based column order): ";
        for (int i = 0; i < keyLength; ++i) {
            cin >> key[i];
        }
        cout << "Encrypt(1) or Decrypt(2): ";
        int action;
        cin >> action;
        string text;
        cout << "Enter text (no spaces): ";
        cin >> text;
        if (action == 1) {
            string encryptedText = colEn(text, key);
            cout << "Encrypted Text: " << encryptedText << endl;
        } else if (action == 2) {
            string decryptedText = colDec(text, key);
            cout << "Decrypted Text: " << decryptedText << endl;
        }
    }
    return 0;
}
```