INFORMATION SECURITY

ITD03

DEPARTMENT OF INFORMATION TECHNOLOGY

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

SHIVANI GUPTA
2018UIT2586
IT-2

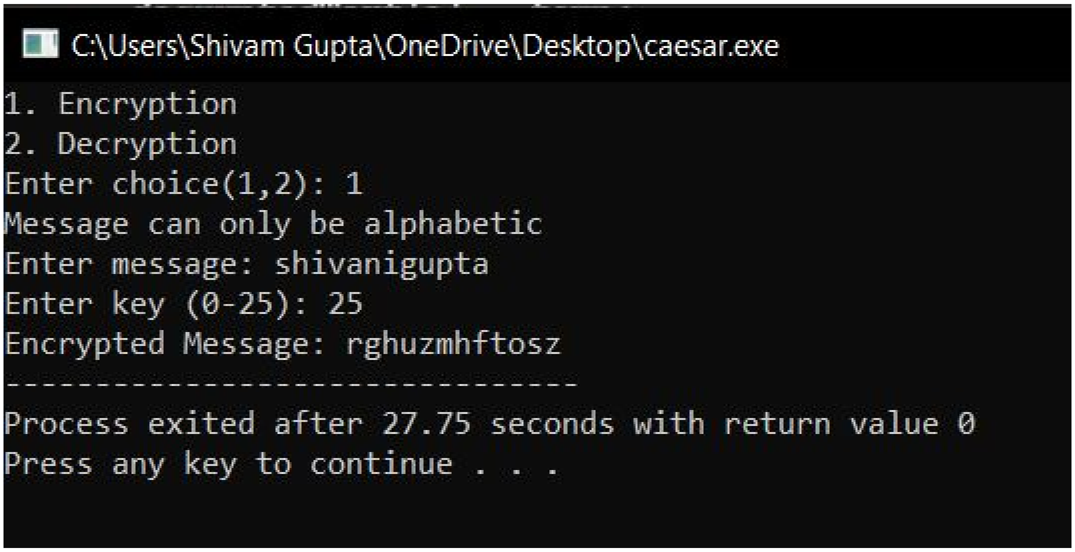
INDEX

CAESAR CIPHER :

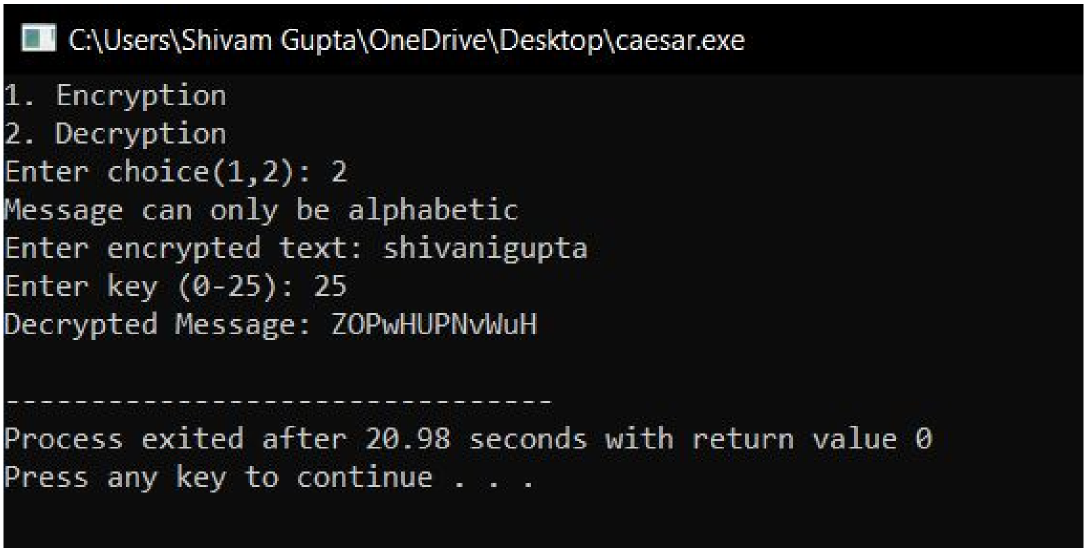
Code:

```
#include <iostream>
using namespace std;
int main()
{
    int choice;
    cout << "1. Encryption" << endl << "2. Decryption" << endl << "Enter
choice(1,2): ";
    cin >> choice;
    cin.ignore();

    if (choice == 1){
        // encryption
        string msg;
        cout << "Message can only be alphabetic" << endl;
        cout << "Enter message: ";
        getline(cin, msg);

        int key;
        cout << "Enter key (0-25): ";
        cin >> key;
        cin.ignore();

        string encryptedText = msg;

        for (int i = 0; i < msg.size(); i ++){
            if(msg[i]==32){
                continue; //32 is ASCII of space character, we will ignore
            } else {
                if((msg[i]+key) > 122) {
                    //after lowercase z move back to a, z's ASCII is 122
                    int temp = (msg[i] + key) - 122;
                    encryptedText[i] = 96 + temp;
                } else if (msg[i] + key > 90 && msg[i] <= 96){
                    //after uppercase Z move back to A, 90 is Z's ASCII
                    int temp = (msg[i] + key) - 90;
                    encryptedText[i] = 64 + temp;
                } else {
                    //in case of characters being in between A-Z & a-z
                    encryptedText[i] += key;
                }
            } //if
        }
        cout << "Encrypted Message: " << encryptedText;

    } else if (choice == 2){
        //decryption
        string encpMsg;
        cout << "Message can only be alphabetic" << endl;
        cout << "Enter encrypted text: ";
        getline(cin, encpMsg);

        int dcyptKey;
        cout << "Enter key (0-25): ";
        cin >> dcyptKey;
        cin.ignore();

        string decryptedText = encpMsg;

        for (int i = 0; i < encpMsg.size(); i++){
            if(encpMsg[i]==32){
                continue; //ignoring space
            } else {
                if((encpMsg[i] - dcyptKey) < 97 && (encpMsg[i] - dcyptKey)
> 90){
                    int temp = (encpMsg[i] - dcyptKey) + 26;
                    decryptedText[i] = temp;
                } else if((encpMsg[i] - dcyptKey) < 65){
                    int temp = (encpMsg[i] - dcyptKey) + 26;
                    decryptedText[i] = temp;
                } else {
                    decryptedText[i] = encpMsg[i] - dcyptKey;
                }
            }
        }
        cout << "Decrypted Message: " << decryptedText << endl;
    } else {
        cout << "Invalid choice";
    }
    return 0;
}
```

OUTPUT:



```
1. Encryption
2. Decryption
Enter choice(1,2): 1
Message can only be alphabetic
Enter message: shivanigupta
Enter key (0-25): 25
Encrypted Message: rghuzmhftosz
--------------------------------
Process exited after 27.75 seconds with return value 0
Press any key to continue . . .
```



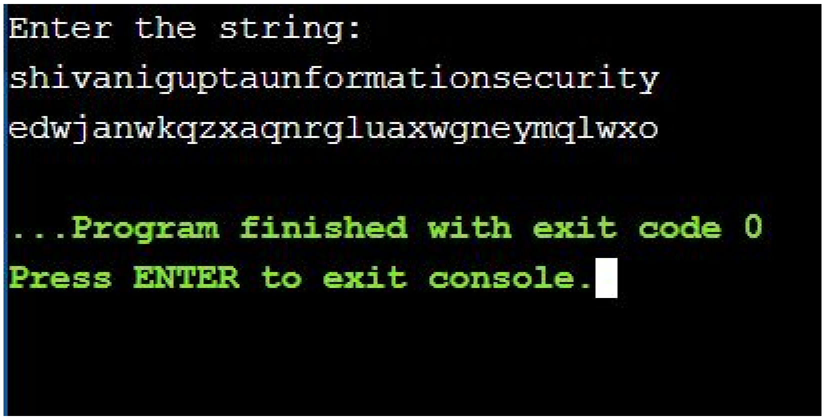
```
1. Encryption
2. Decryption
Enter choice(1,2): 2
Message can only be alphabetic
Enter encrypted text: shivanigupta
Enter key (0-25): 25
Decrypted Message: ZOPwHUPNvWuH

--------------------------------
Process exited after 20.98 seconds with return value 0
Press any key to continue . . .
```

MULTIPLICATIVE CIPHER

Code:

```cpp
#include <iostream>

using namespace std;

int inverse(int a, int m=26)
{
    a = a%m;
    for (int x=1; x<m; x++)
       if ((a*x) % m == 1)
          return x;
}

string encrypt(string plaintext, int k)
{
    string ciphertext;
    for(int i=0; i<plaintext.size(); i++)
    if(plaintext[i]!=' ')
    ciphertext+=((plaintext[i]-97)*k)%26+97;
    else ciphertext+=' ';
    return ciphertext;
}

string decrypt(string ciphertext, int k)
{
    string plaintext;
    for(int i=0; i<ciphertext.size(); i++)
    if(ciphertext[i]!=' ')
    plaintext+=((ciphertext[i]-97)*inverse(k))%26+97;
    else plaintext+=' ';
    return plaintext;
}

int main()
{
    string s;
    cout<<"Enter the string:"<<endl;
    cin>>s;
    int k=11, process=1; //0 for encryption, 1 for decryption
    //cin>>process;
    if(process==0)
    cout<<encrypt(s,k);
    else if(process==1)
    cout<<decrypt(s,k);
    return 0;
}
```

Output:



```
Enter the string:
shivaniguptaunformationsecurity
edwjanwkqzxaqnrgluaxwgneymqlwxo

...Program finished with exit code 0
Press ENTER to exit console.
```

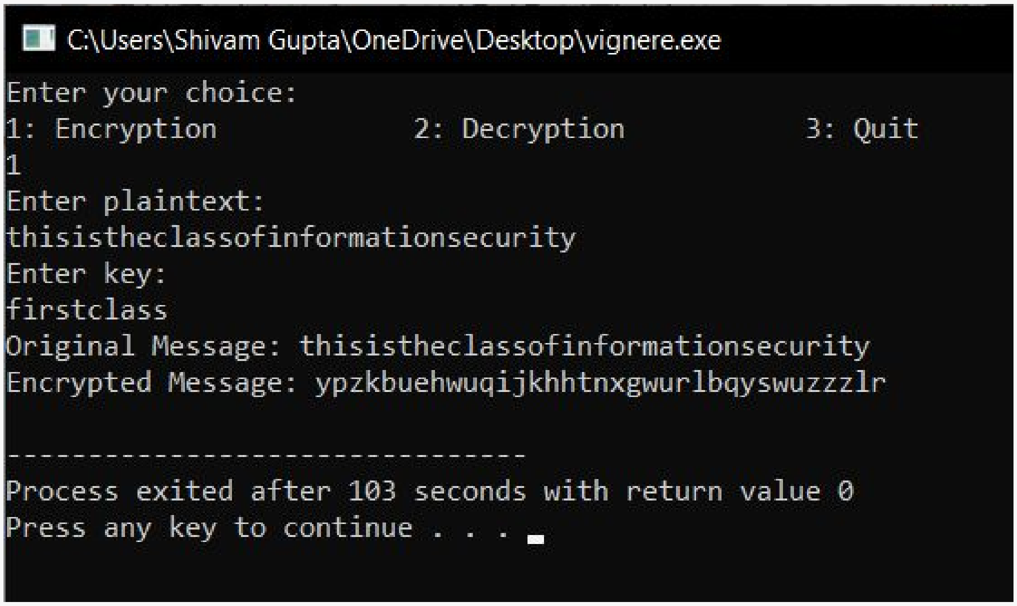
VIGNERE CIPHER

Code:

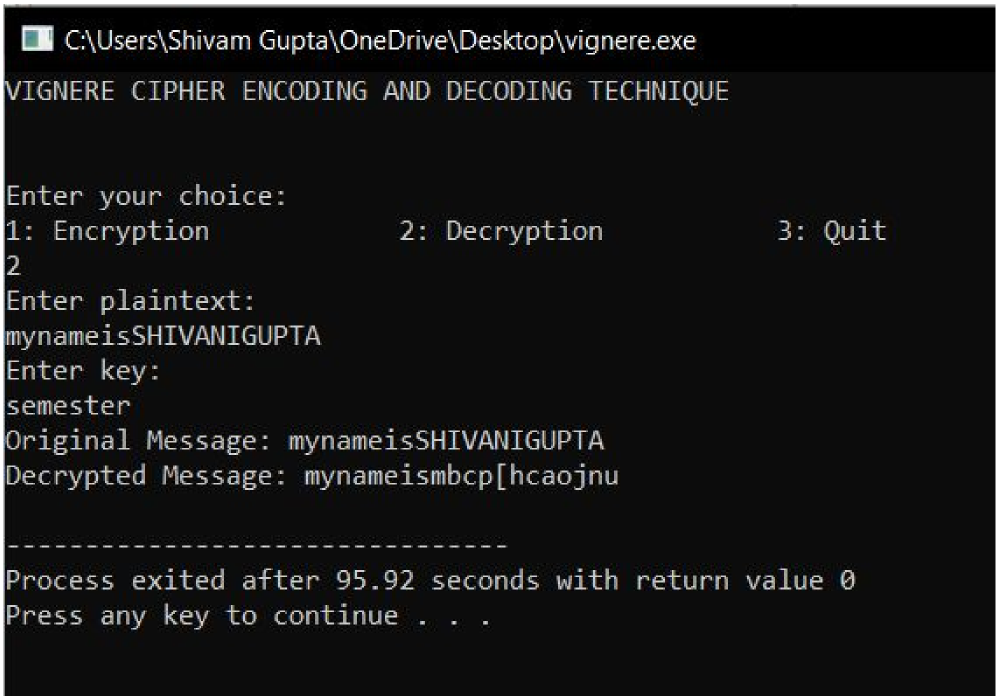
```
#include <iostream>
#include <string>
using namespace std;

   string encryption(string plaintext, string key) {
      string ciphertext;
      for (int i = 0; i < plaintext.length(); ++i) {
         char c = plaintext[i];
         if(c!=' '){
             ciphertext += (((plaintext[i]-97) + (key[i%key.length()]-97))%26)+97;
         }
         else{
             ciphertext = " ";
         }
      }
      return ciphertext;
   }
   string decryption(string ciphertext, string key) {
      string plaintext;
      for(int i=0;i<ciphertext.length();i++){
          if(ciphertext[i]!=' '){
              plaintext += (((ciphertext[i]-97)+(26-(key[i%key.length()]-97)))%26)+97;
          }
          else{
              plaintext = " ";
          }
      }
      return plaintext;
   }

int main() {
   string plaintext, key, encrypt, decrypt;
   cout<<"Enter your choice: \n";
```

```
    cout<<"1: Encryption \t\t 2: Decryption \t\t 3: Quit\n";
    int choice;
    cin>>choice;
    if(choice==3)
    return 0;
    cout<<"Enter plaintext: \n";
    cin>>plaintext;
    cout<<"Enter key: \n";
    cin>>key;
    if(choice==1){
    encrypt = encryption(plaintext, key);
    cout << "Original Message: "<< plaintext << endl;
    cout << "Encrypted Message: " << encrypt << endl;
    }
    else if(choice==2){
    decrypt = decryption(encrypt, key);
     cout << "Original Message: "<< plaintext << endl;
    cout << "Decrypted Message: " << decrypt << endl;
    }
}
```

Output:



```
Enter your choice:
1: Encryption            2: Decryption           3: Quit
1
Enter plaintext:
thisistheclassofinformationsecurity
Enter key:
firstclass
Original Message: thisistheclassofinformationsecurity
Encrypted Message: ypzkbuehwuqijkhhtnxgwurlbqyswuzzzlr

--------------------------------
Process exited after 103 seconds with return value 0
Press any key to continue . . .
```



```
VIGNERE CIPHER ENCODING AND DECODING TECHNIQUE


Enter your choice:
1: Encryption            2: Decryption           3: Quit
2
Enter plaintext:
mynameisSHIVANIGUPTA
Enter key:
semester
Original Message: mynameisSHIVANIGUPTA
Decrypted Message: mynameismbcp[hcaojnu

--------------------------------
Process exited after 95.92 seconds with return value 0
Press any key to continue . . .
```

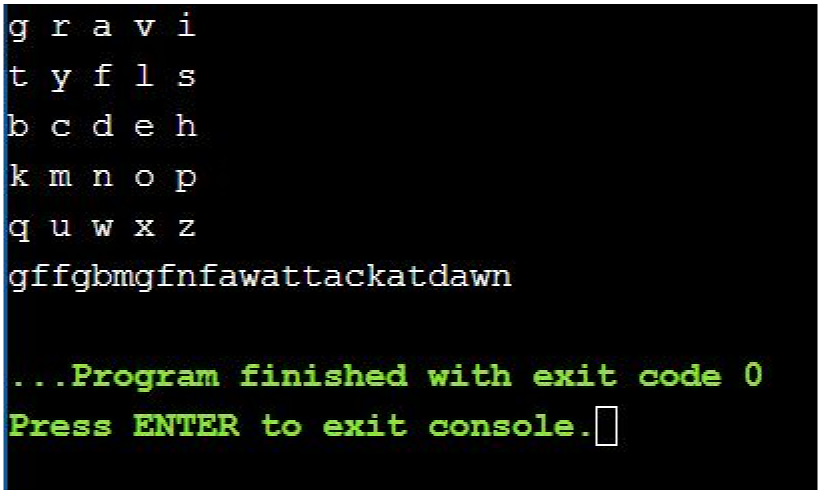
PLAYFAIR CIPHER

Code:

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

pair<int, int> getposition(vector<vector<char>> key, char c){
    for(int i=0; i<5; i++)
        for(int j=0; j<5; j++)
            if(key[i][j]==c)
                return {i,j};
}

string encrypt(vector<vector<char>> key, string plaintext){
    string encrypted;
    char a,b;
    int i=0;
    while(i<plaintext.size()){
        a=plaintext[i];
        if(i==plaintext.size()-1 || (plaintext[i]==plaintext[i+1])){
            b='x';
            i++;
        }
        else{
            b=plaintext[i+1];
            i+=2;
            }
        pair<int, int> positiona=getposition(key, a),
positionb=getposition(key, b);
        if(positiona.first==positionb.first){
                encrypted+=key[positiona.first][(positiona.second+1)%5];
                encrypted+=key[positionb.first][(positionb.second+1)%5];
        }
        else if(positiona.second==positionb.second){
                encrypted+=key[(positiona.first+1)%5][positiona.second];
                encrypted+=key[(positionb.first+1)%5][positionb.second];
        }
        else{
                encrypted+=key[positiona.first][positionb.second];
                encrypted+=key[positionb.first][positiona.second];
        }
    }
    return encrypted;
}

string decrypt(vector<vector<char>> key, string ciphertext){
    string decrypted;
    char a,b;
    int i=0;
    while(i<ciphertext.size()){
        a=ciphertext[i];
        if(i==ciphertext.size()-1 || (ciphertext[i]==ciphertext[i+1])){
            b='x';
            i++;
        }
        else{
            b=ciphertext[i+1];
            i+=2;
            }
        pair<int, int> positiona=getposition(key, a),
positionb=getposition(key, b);
        if(positiona.first==positionb.first){
                if(positiona.second==0)
                    positiona.second=4;
                else positiona.second--;
                if(positionb.second==0)
                    positionb.second=4;
                else positionb.second--;
                decrypted+=key[positiona.first][positiona.second];
                decrypted+=key[positionb.first][positionb.second];
        }
        else if(positiona.second==positionb.second){
                if(positiona.first==0)
                    positiona.first=4;
                else positiona.first--;
                if(positionb.first==0)
```

```
                    positionb.first=4;
                else positionb.first--;
                decrypted+=key[positiona.first][positiona.second];
                decrypted+=key[positionb.first][positionb.second];
        }
        else{
                decrypted+=key[positiona.first][positionb.second];
                decrypted+=key[positionb.first][positiona.second];
        }
    }
    return decrypted;
}

void fillKeyMatrix(vector<vector<char>> &key, string keyword){
    keyword.append("abcdefghijklmnopqrstuvwxyz");
    vector<bool> filled(26, false);
    int index=0;
    for(int i=0; i<keyword.size(); i++){
            if(filled[keyword[i]-97]==false){
                key[index/5][index%5]=keyword[i];
                filled[keyword[i]-97]=true;
                if(keyword[i]=='i' || keyword[i]=='j'){
                    filled['i'-97]=true;
                    filled['j'-97]=true;
                }
                index++;
                }
    }
}

int main(){
    vector<vector<char>> key(5, vector<char>(5));
    fillKeyMatrix(key, "gravityfalls");
    for(int i=0; i<5; i++)
    {   for(int j=0; j<5; j++)
            cout<<key[i][j]<<' ';
        cout<<'\n';
    }
    cout<<encrypt(key,"attackatdawn");
    cout<<decrypt(key,"gffgbmgfnfaw");
}
```

Output:

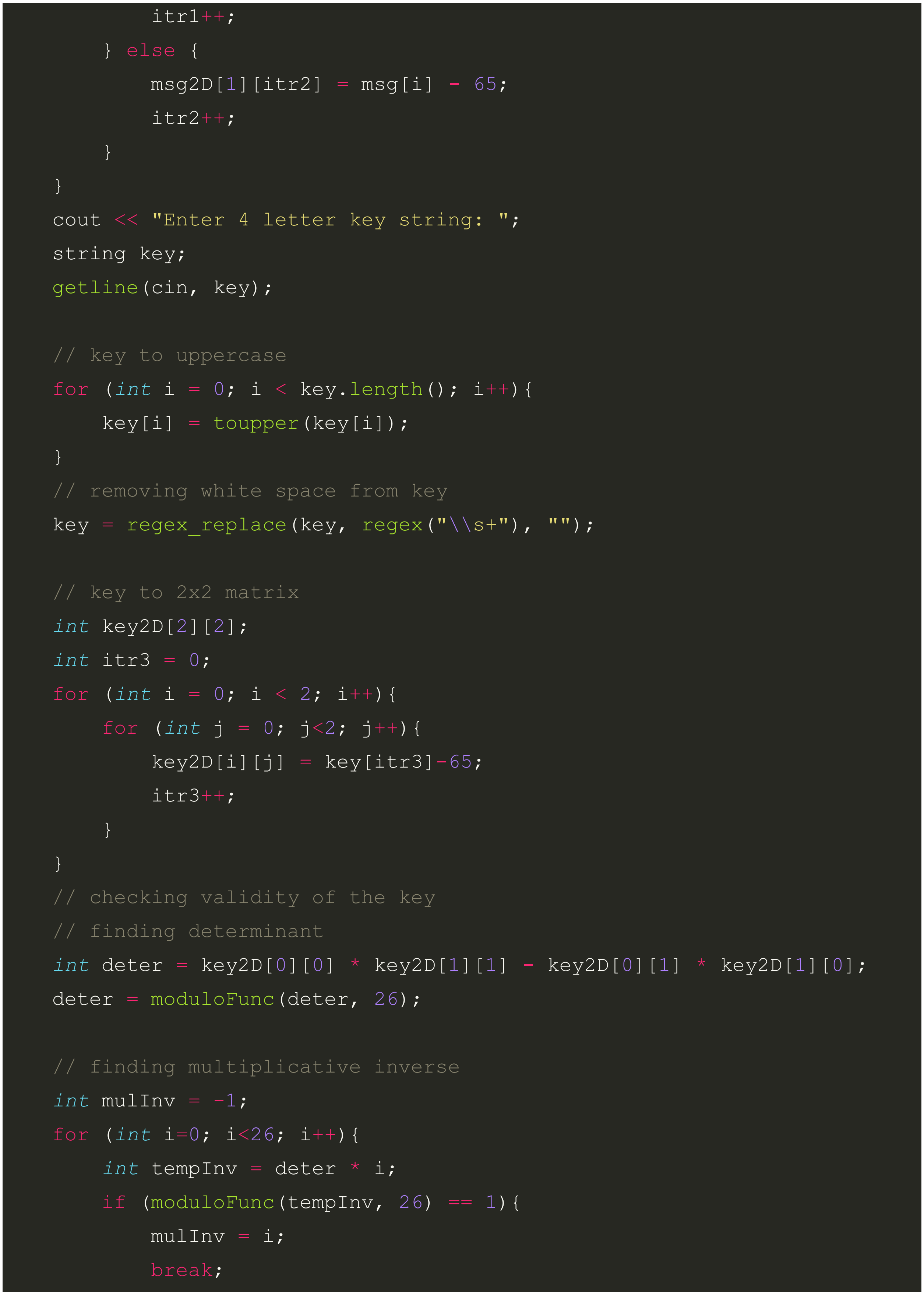


```
g r a v i
t y f l s
b c d e h
k m n o p
q u w x z
gffgbmgfnfawattackatdawn

...Program finished with exit code 0
Press ENTER to exit console.
```

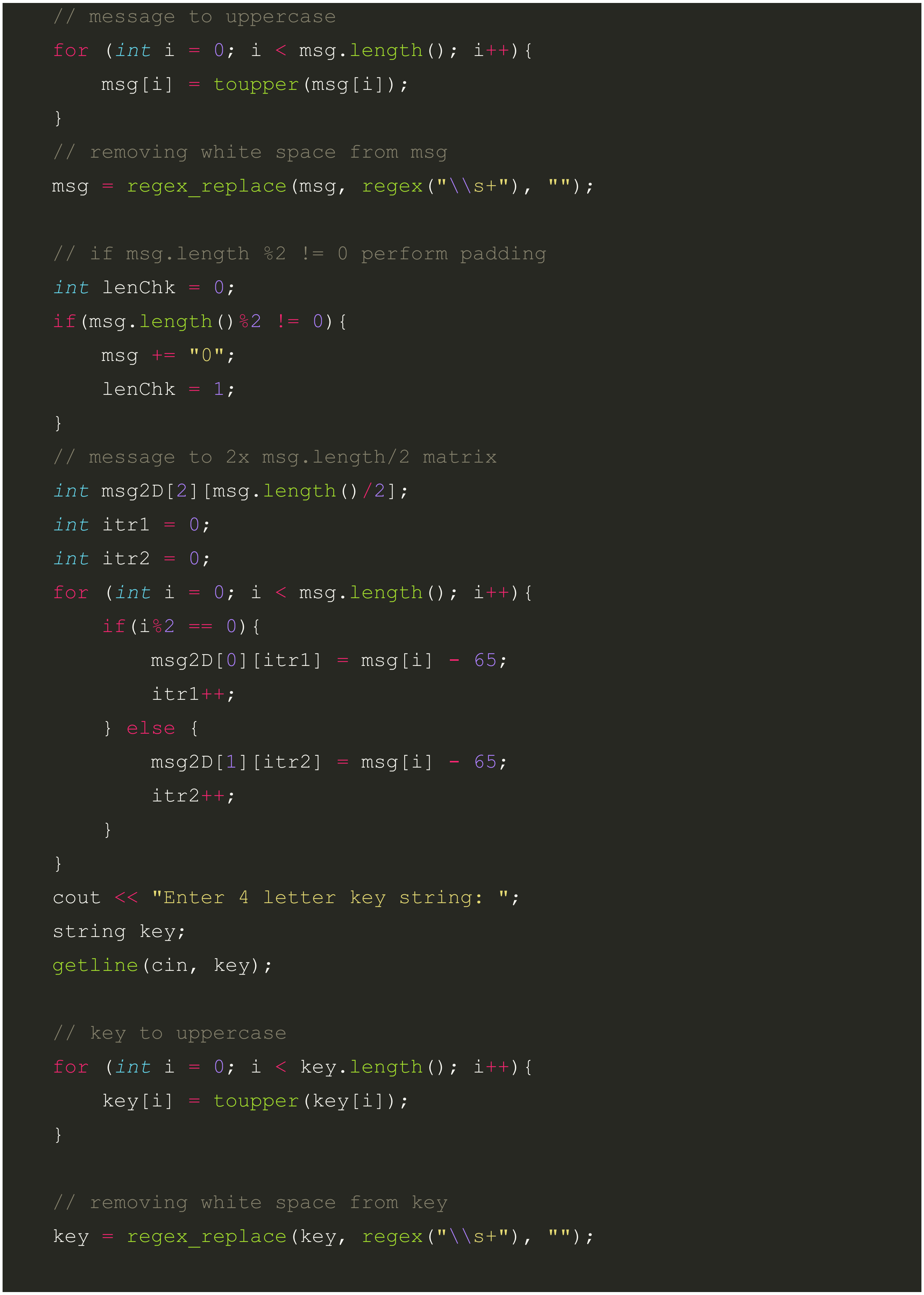
HILL CIPHER

Code:

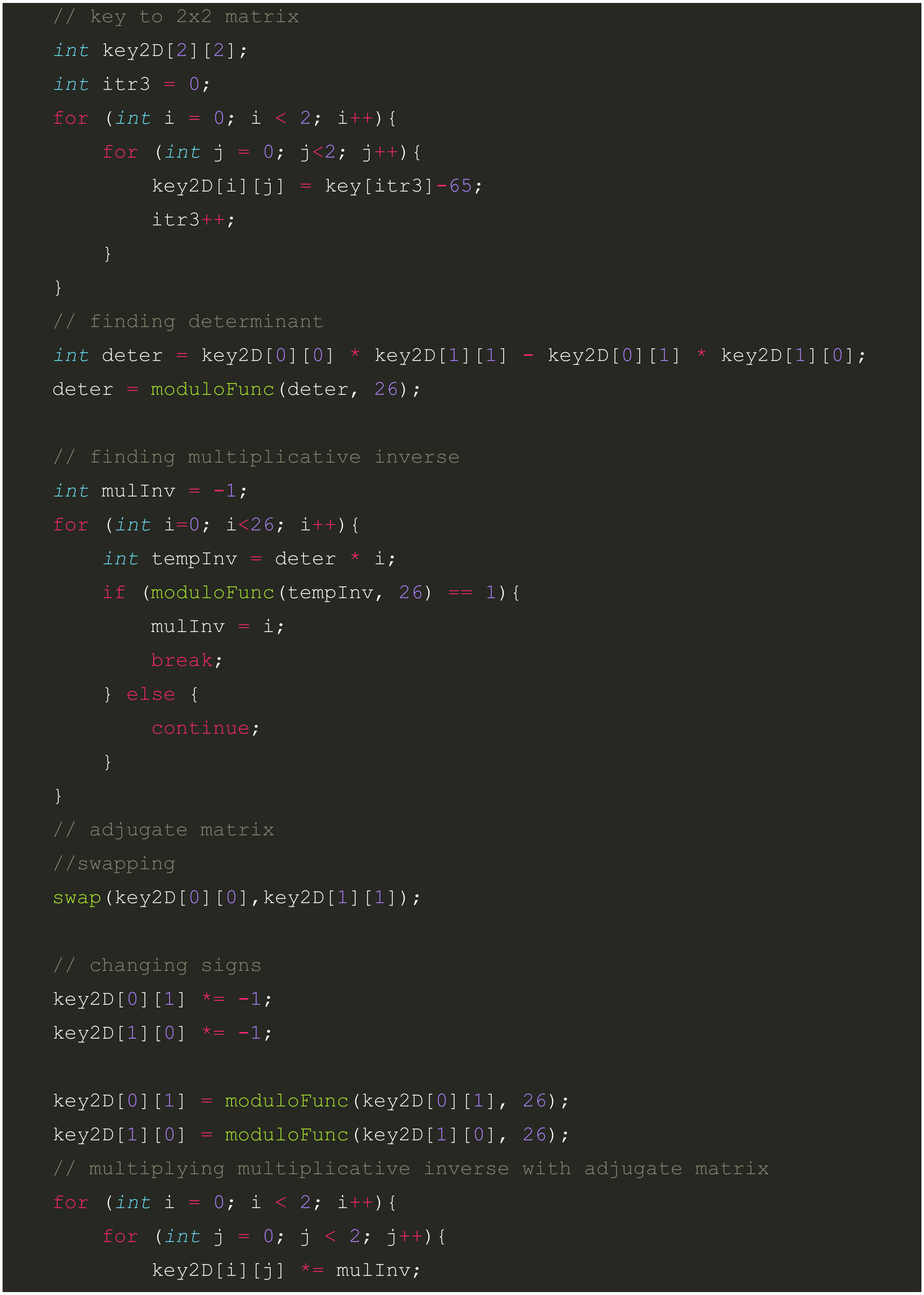
```
#include <bits/stdc++.h>
#include <regex>
using namespace std;

int moduloFunc(int a, int b){
    int result = a % b;
    if (result < 0){
        result += b;
    }
    return result;
}
void cipherEncryption(){
    string msg;
    cout << "Enter message: ";
    getline(cin, msg);

    // message to uppercase
    for (int i = 0; i < msg.length(); i++){
        msg[i] = toupper(msg[i]);
    }
    // removing white space from msg
    msg = regex_replace(msg, regex("\\s+"), "");

    // if msg.length %2 != 0 perform padding
    int lenChk = 0;
    if(msg.length()%2 != 0){
        msg += "0";
        lenChk = 1;
    }
    // message to 2x msg.length/2 matrix
    int msg2D[2][msg.length()/2];
    int itr1 = 0;
    int itr2 = 0;
    for (int i = 0; i < msg.length(); i++){
        if(i%2 == 0){
            msg2D[0][itr1] = msg[i] - 65;
```



```
        itr1++;
    } else {
        msg2D[1][itr2] = msg[i] - 65;
        itr2++;
    }
}
cout << "Enter 4 letter key string: ";
string key;
getline(cin, key);

// key to uppercase
for (int i = 0; i < key.length(); i++){
    key[i] = toupper(key[i]);
}
// removing white space from key
key = regex_replace(key, regex("\\s+"), "");

// key to 2x2 matrix
int key2D[2][2];
int itr3 = 0;
for (int i = 0; i < 2; i++){
    for (int j = 0; j<2; j++){
        key2D[i][j] = key[itr3]-65;
        itr3++;
    }
}
// checking validity of the key
// finding determinant
int deter = key2D[0][0] * key2D[1][1] - key2D[0][1] * key2D[1][0];
deter = moduloFunc(deter, 26);

// finding multiplicative inverse
int mulInv = -1;
for (int i=0; i<26; i++){
    int tempInv = deter * i;
    if (moduloFunc(tempInv, 26) == 1){
        mulInv = i;
        break;
        } else {
            continue;
        }
    }
    if (mulInv == -1){
        cout << "invalid key" << endl;
        exit(EXIT_FAILURE);
    }

    string encrypText = "";
    int itrCount = msg.length()/2;
    if (lenChk == 0){
        // if msg.length % 2 == 0
        for (int i = 0; i < itrCount; i++){
            int temp1 = msg2D[0][i] * key2D[0][0] + msg2D[1][i] *
key2D[0][1];
            encrypText += (char)((temp1 % 26) + 65);
            int temp2 = msg2D[0][i] * key2D[1][0] + msg2D[1][i] *
key2D[1][1];
            encrypText += (char)((temp2 % 26) + 65);
        }
    } else {
        // if msg.length % 2 == 0
        for (int i = 0; i < itrCount-1; i++){
            int temp1 = msg2D[0][i] * key2D[0][0] + msg2D[1][i] *
key2D[0][1];
            encrypText += (char)((temp1 % 26) + 65);
            int temp2 = msg2D[0][i] * key2D[1][0] + msg2D[1][i] *
key2D[1][1];
            encrypText += (char)((temp2 % 26) + 65);
        }
    }
    cout << endl << "Encrypted text: " << encrypText << endl;
}
void cipherDecryption(){
    string msg;
    cout << "Enter message: ";
    getline(cin, msg);
```

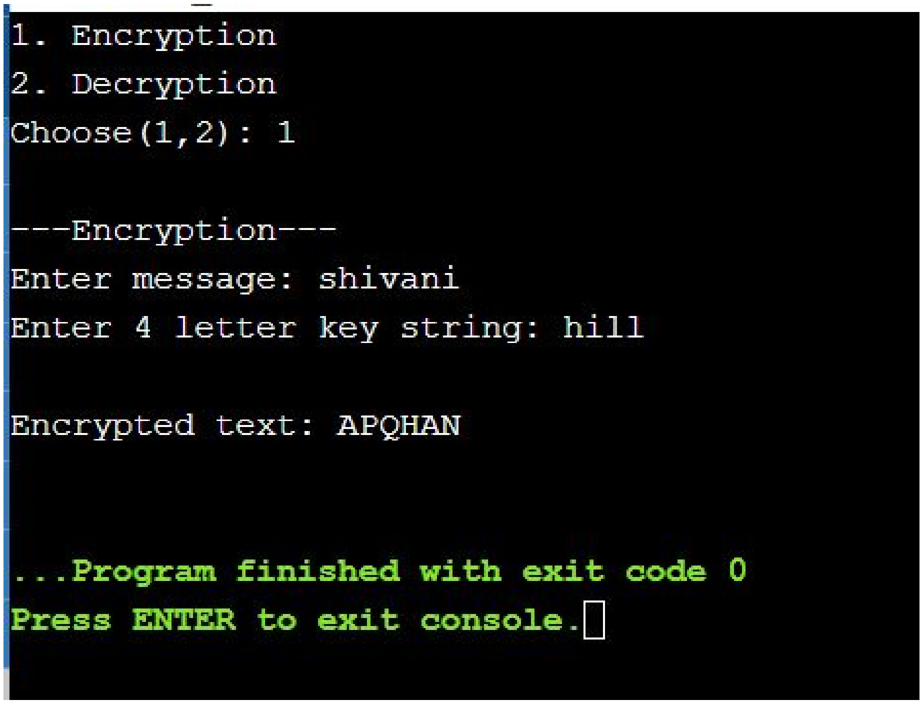


```
// message to uppercase
for (int i = 0; i < msg.length(); i++){
    msg[i] = toupper(msg[i]);
}
// removing white space from msg
msg = regex_replace(msg, regex("\\s+"), "");

// if msg.length %2 != 0 perform padding
int lenChk = 0;
if(msg.length()%2 != 0){
    msg += "0";
    lenChk = 1;
}
// message to 2x msg.length/2 matrix
int msg2D[2][msg.length()/2];
int itr1 = 0;
int itr2 = 0;
for (int i = 0; i < msg.length(); i++){
    if(i%2 == 0){
        msg2D[0][itr1] = msg[i] - 65;
        itr1++;
    } else {
        msg2D[1][itr2] = msg[i] - 65;
        itr2++;
    }
}
cout << "Enter 4 letter key string: ";
string key;
getline(cin, key);

// key to uppercase
for (int i = 0; i < key.length(); i++){
    key[i] = toupper(key[i]);
}

// removing white space from key
key = regex_replace(key, regex("\\s+"), "");
```



```
// key to 2x2 matrix
int key2D[2][2];
int itr3 = 0;
for (int i = 0; i < 2; i++){
    for (int j = 0; j<2; j++){
        key2D[i][j] = key[itr3]-65;
        itr3++;
    }
}
// finding determinant
int deter = key2D[0][0] * key2D[1][1] - key2D[0][1] * key2D[1][0];
deter = moduloFunc(deter, 26);

// finding multiplicative inverse
int mulInv = -1;
for (int i=0; i<26; i++){
    int tempInv = deter * i;
    if (moduloFunc(tempInv, 26) == 1){
        mulInv = i;
        break;
    } else {
        continue;
    }
}
// adjugate matrix
//swapping
swap(key2D[0][0],key2D[1][1]);

// changing signs
key2D[0][1] *= -1;
key2D[1][0] *= -1;

key2D[0][1] = moduloFunc(key2D[0][1], 26);
key2D[1][0] = moduloFunc(key2D[1][0], 26);
// multiplying multiplicative inverse with adjugate matrix
for (int i = 0; i < 2; i++){
    for (int j = 0; j < 2; j++){
        key2D[i][j] *= mulInv;
        }
    }
    for (int i=0; i<2;i++){
        for(int j =0; j <2; j++){
            key2D[i][j] = moduloFunc(key2D[i][j], 26);
        }
    }
    // ciphertext to plain text
    string decrypText = "";
    int itrCount = msg.length()/2;
    if (lenChk == 0){
        // if msg.length % 2 == 0
        for (int i = 0; i < itrCount; i++){
            int temp1 = msg2D[0][i] * key2D[0][0] + msg2D[1][i] *
key2D[0][1];
            decrypText += (char)((temp1 % 26) + 65);
            int temp2 = msg2D[0][i] * key2D[1][0] + msg2D[1][i] *
key2D[1][1];
            decrypText += (char)((temp2 % 26) + 65);
        }
    } else {
        // if msg.length % 2 == 0
        for (int i = 0; i < itrCount-1; i++){
            int temp1 = msg2D[0][i] * key2D[0][0] + msg2D[1][i] *
key2D[0][1];
            decrypText += (char)((temp1 % 26) + 65);
            int temp2 = msg2D[0][i] * key2D[1][0] + msg2D[1][i] *
key2D[1][1];
            decrypText += (char)((temp2 % 26) + 65);
        }
    }
    cout << endl << "Decrypted text: " << decrypText << endl;
}
int main()
{
    cout << "1. Encryption\n2. Decryption\nChoose(1,2): ";
    int choice;
    cin >> choice;
    cin.ignore();
    if (choice == 1){
        cout << endl << "---Encryption---" << endl;
        cipherEncryption();
    } else if (choice == 2){
        cout << endl << "---Decryption---" << endl;
        cipherDecryption();
    } else {
        cout << endl << "Wrong choice" << endl;
    }
    return 0;
}
```

OUTPUT:



```
1. Encryption
2. Decryption
Choose(1,2): 1

---Encryption---
Enter message: shivani
Enter 4 letter key string: hill

Encrypted text: APQHAN


...Program finished with exit code 0
Press ENTER to exit console.
```

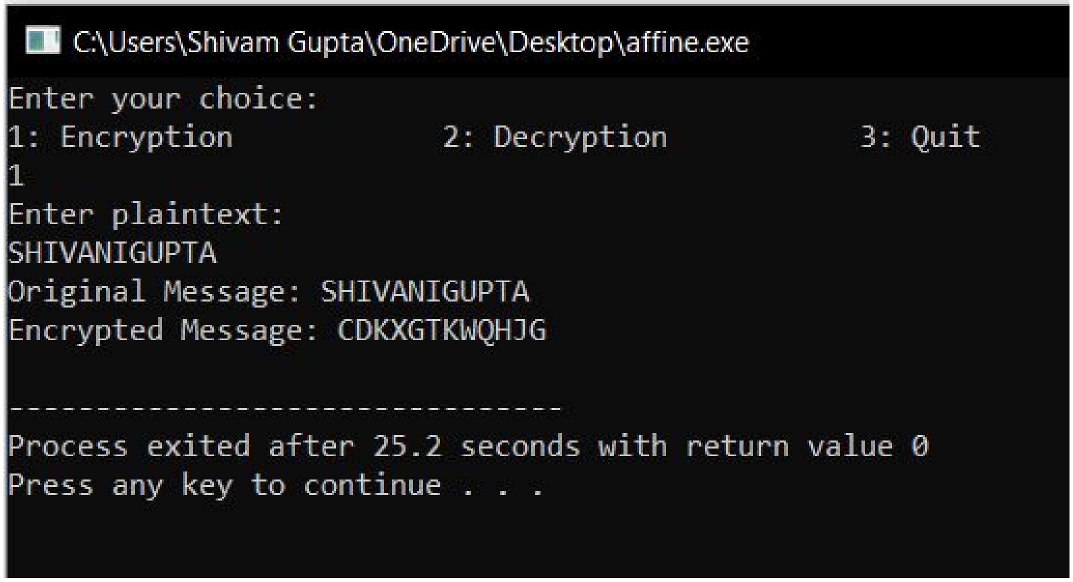
AFFINE CIPHER

CODE:

```
#include<bits/stdc++.h>
using namespace std;
static int a = 7;
static int b = 6;
string encryption(string m) {
    //Cipher Text initially empty
    string c = "";
    for (int i = 0; i < m.length(); i++) {
        // Avoid space to be encrypted
        if(m[i]!=' ')
            // added 'A' to bring it in range of ASCII alphabet [ 65-90 | A-Z ]
            c = c + (char) ((((a * (m[i]-'A') ) + b) % 26) + 'A');
        else
            //else append space character
            c += m[i];
    }
    return c;
}
string decryption(string c) {
    string m = "";
    int a_inverse = 0;
    int flag = 0;
    //Find a^-1 (the multiplicative inverse of a
    //in the group of integers modulo m.)
    for (int i = 0; i < 26; i++) {
        flag = (a * i) % 26;
        //Check if (a * i) % 26 == 1,
        //then i will be the multiplicative inverse of a
        if (flag == 1) {
            a_inverse = i;
        }
    }
    for (int i = 0; i < c.length(); i++) {
        if(c[i] != ' ')
```

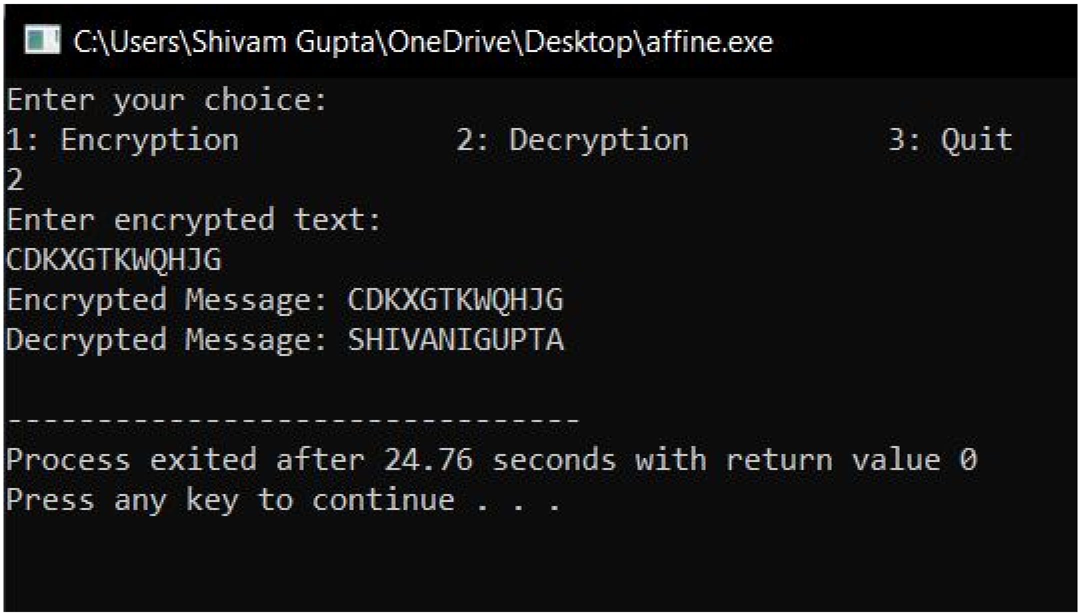
```
         // added 'A' to bring it in range of ASCII alphabet [ 65-90 | A-Z ]
         m = m + (char) (((a_inverse * ((c[i]+'A' - b)) % 26)) + 'A');
      else
         //else append space character
         m += c[i];
   }
   return m;
}
int main(void) {
    string plaintext, encrypt, decrypt;
     cout<<"Enter your choice: \n";
   cout<<"1: Encryption \t\t 2: Decryption \t\t 3: Quit\n";
   int choice;
   cin>>choice;
   if(choice==3)
   return 0;
   if(choice==1){
     cout<<"Enter plaintext: \n";
   cin>>plaintext;
   encrypt = encryption(plaintext);
   cout << "Original Message: "<< plaintext << endl;
   cout << "Encrypted Message: " << encrypt << endl;
   }
   else if(choice==2){
        cout<<"Enter encrypted text: \n";
   cin>>encrypt;
   decrypt = decryption(encrypt);
    cout << "Encrypted Message: "<< encrypt << endl;
   cout << "Decrypted Message: " << decrypt << endl;
   }
}
```

OUTPUT:



C:\Users\Shivam Gupta\OneDrive\Desktop\affine.exe

```
Enter your choice:
1: Encryption            2: Decryption           3: Quit
1
Enter plaintext:
SHIVANIGUPTA
Original Message: SHIVANIGUPTA
Encrypted Message: CDKXGTKWQHJG

--------------------------------
Process exited after 25.2 seconds with return value 0
Press any key to continue . . .
```



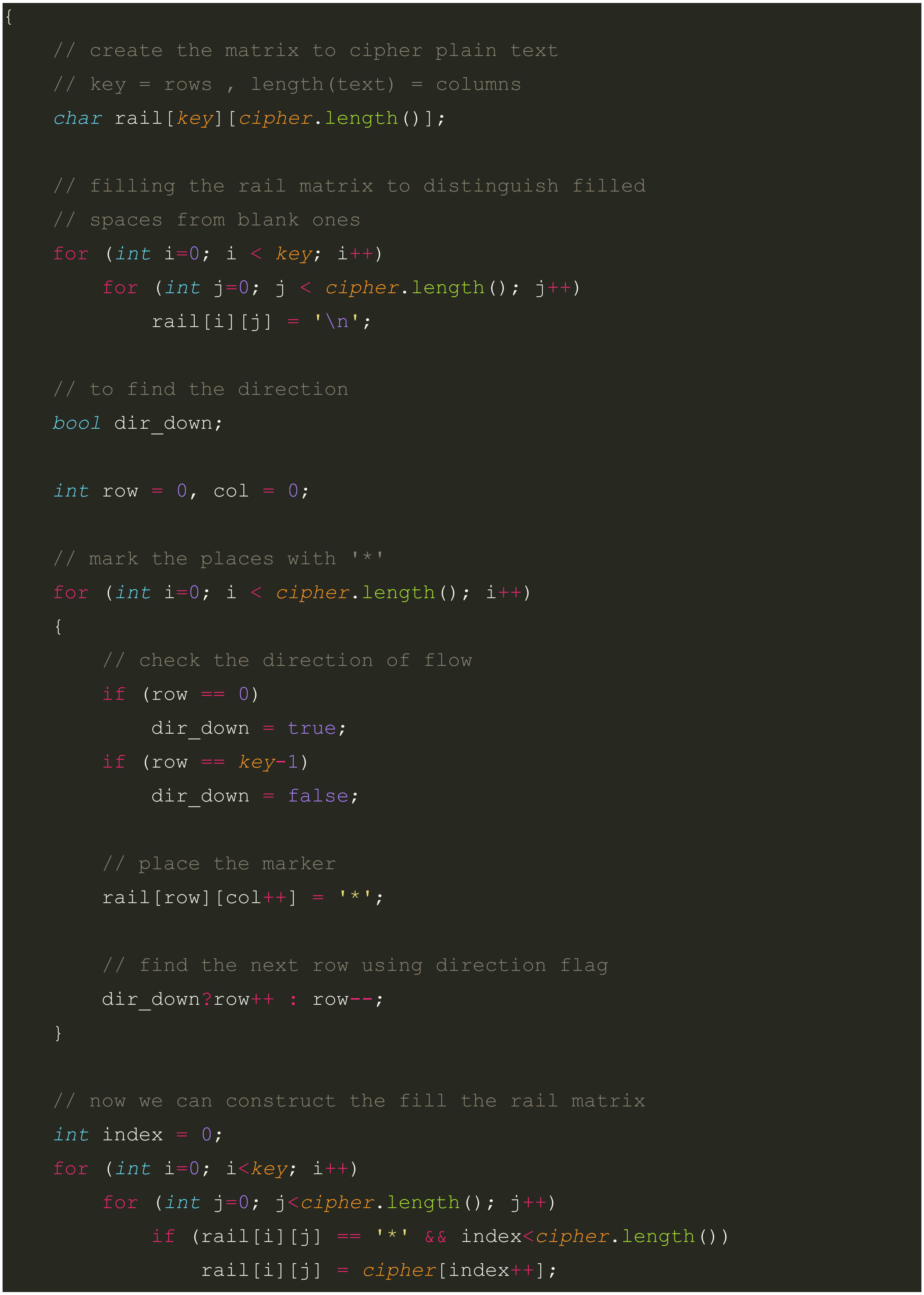
C:\Users\Shivam Gupta\OneDrive\Desktop\affine.exe

```
Enter your choice:
1: Encryption            2: Decryption           3: Quit
2
Enter encrypted text:
CDKXGTKWQHJG
Encrypted Message: CDKXGTKWQHJG
Decrypted Message: SHIVANIGUPTA

--------------------------------
Process exited after 24.76 seconds with return value 0
Press any key to continue . . .
```

RAIL FENCE-ROW AND COLUMN TRANSFORMATION

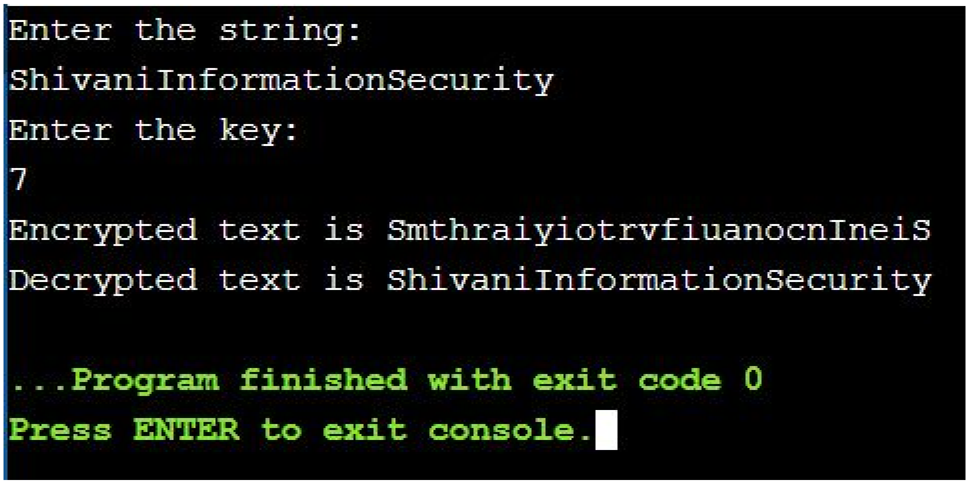
Code:

```
#include <bits/stdc++.h>
using namespace std;

// encryption
string encryptRailFence(string text, int key)
{
    char rail[key][(text.length())];
    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++)
    {
        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;
}
// 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++];
    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;
}
int main()
{
    string str;
    cout<<"Enter the string:"<<endl;
    cin>>str;
    cout<<"Enter the key:"<<endl;
    int key;
    cin>>key;
    string enc=encryptRailFence(str, key);
    cout<<"Encrypted text is "<<enc<<endl;
    cout<<"Decrypted text is "<<decryptRailFence(enc,key);
    return 0;
}
```

OUTPUT:

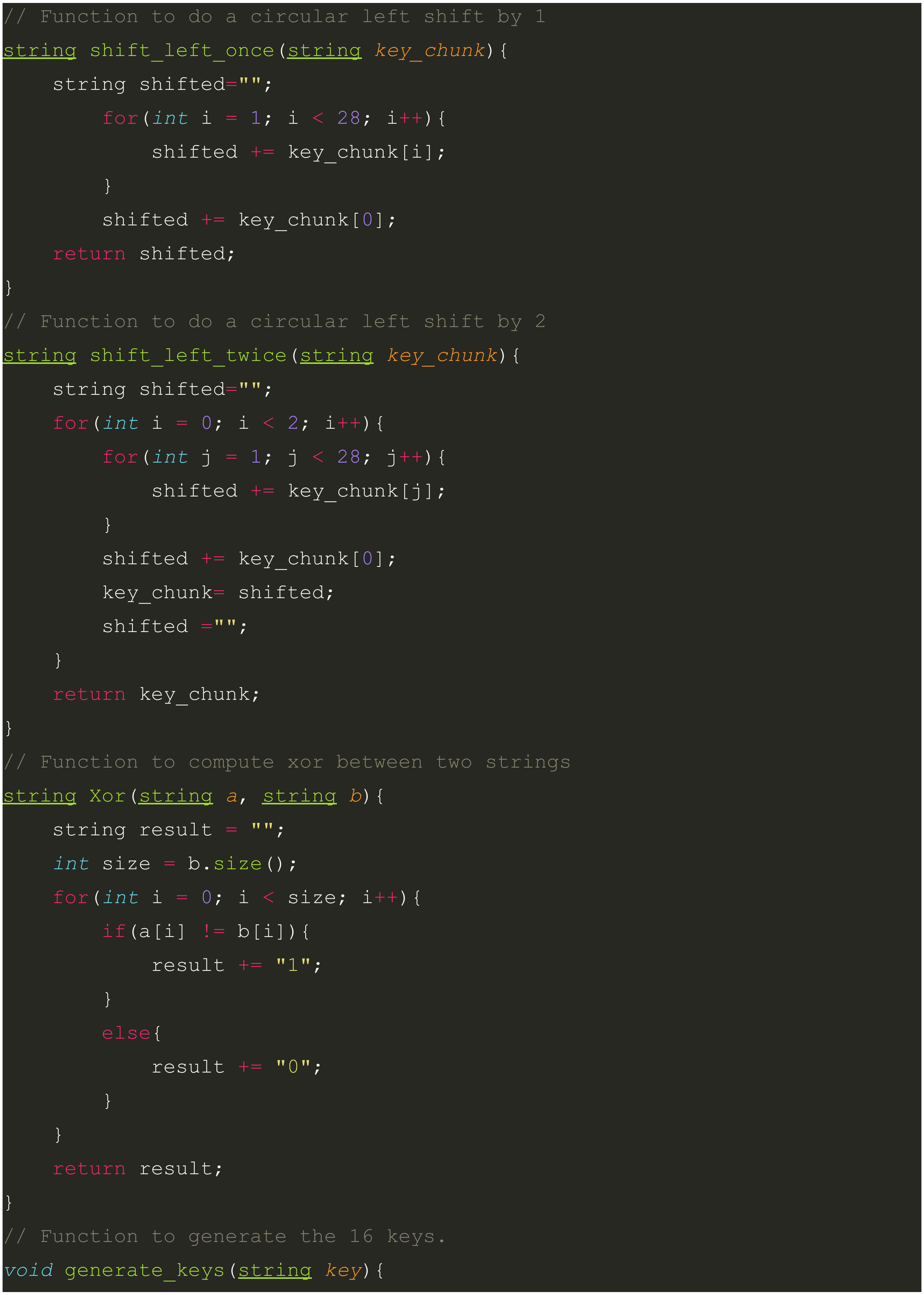


```
Enter the string:
ShivaniInformationSecurity
Enter the key:
7
Encrypted text is SmthraiyiotrvfiuanocnIneiS
Decrypted text is ShivaniInformationSecurity

...Program finished with exit code 0
Press ENTER to exit console.
```

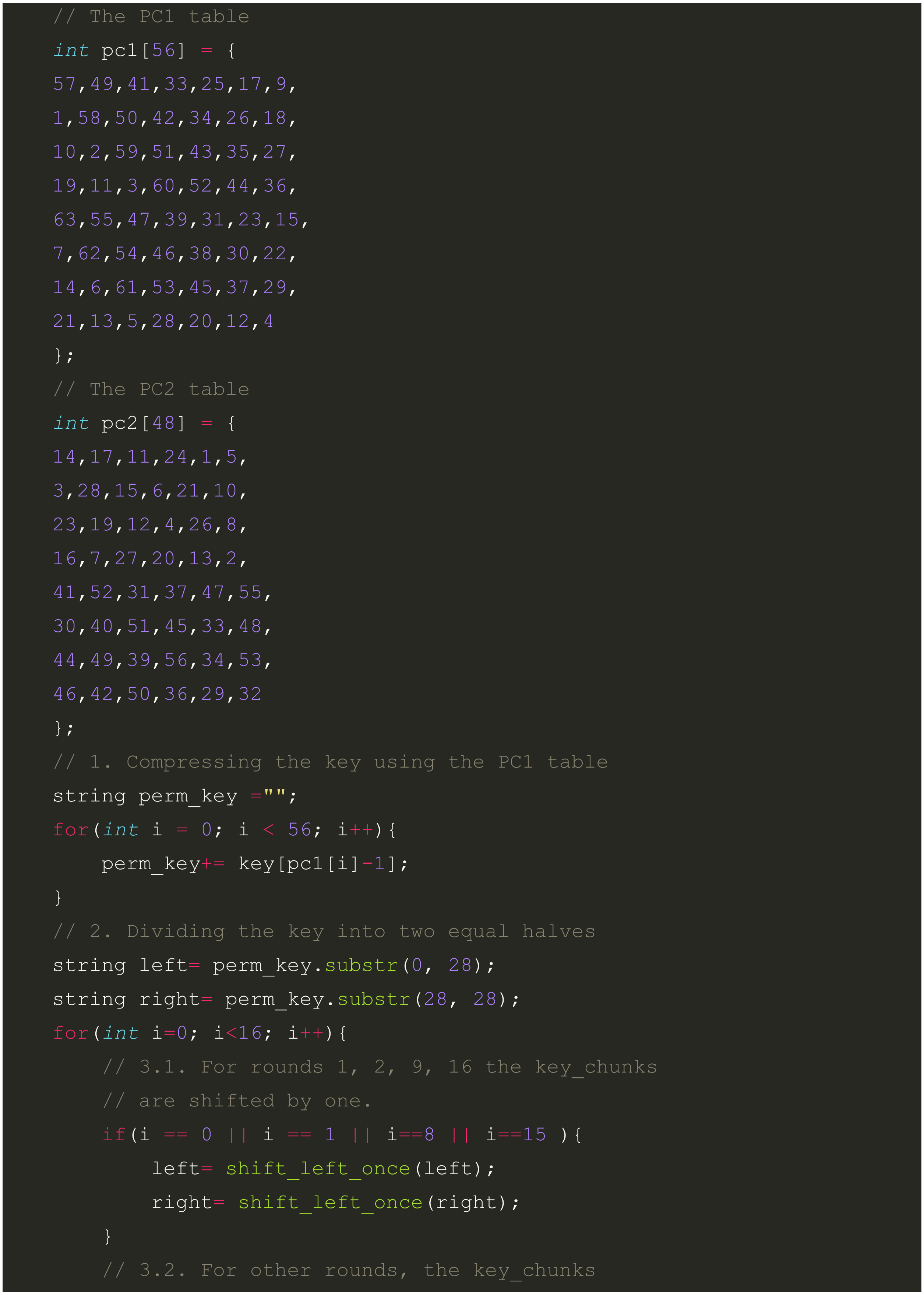
DES

Code:

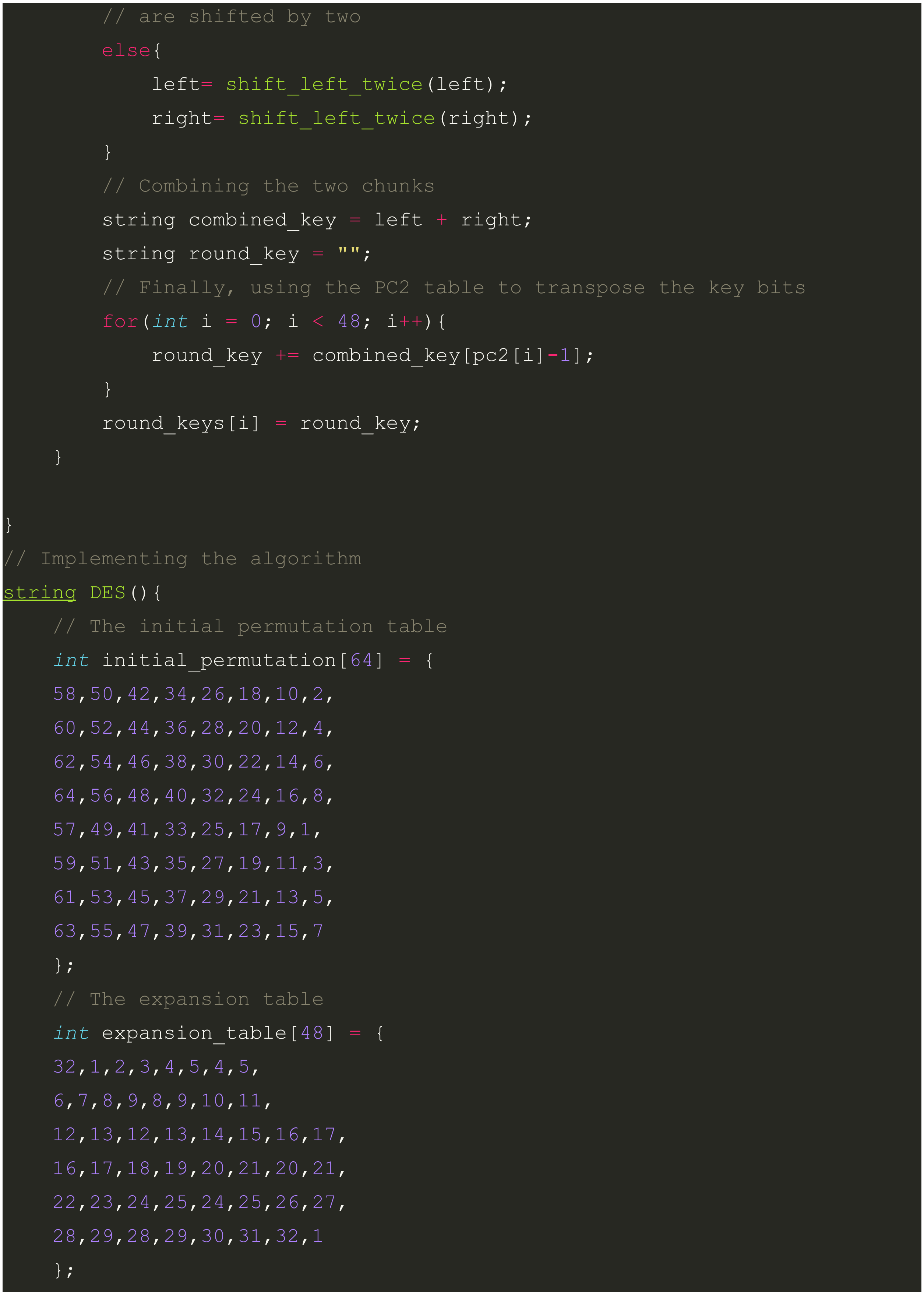
```
#in#include <iostream>
#include <string>
#include <cmath>
using namespace std;
// Array to hold 16 keys
string round_keys[16];
// String to hold the plain text
string pt;
// Function to convert a number in decimal to binary
string convertDecimalToBinary(int decimal)
{
    string binary;
    while(decimal != 0) {
        binary = (decimal % 2 == 0 ? "0" : "1") + binary;
        decimal = decimal/2;
    }
    while(binary.length() < 4){
        binary = "0" + binary;
    }
    return binary;
}
// Function to convert a number in binary to decimal
int convertBinaryToDecimal(string binary)
{
    int decimal = 0;
    int counter = 0;
    int size = binary.length();
    for(int i = size-1; i >= 0; i--)
    {
        if(binary[i] == '1'){
            decimal += pow(2, counter);
        }
    counter++;
    }
    return decimal;
}
```



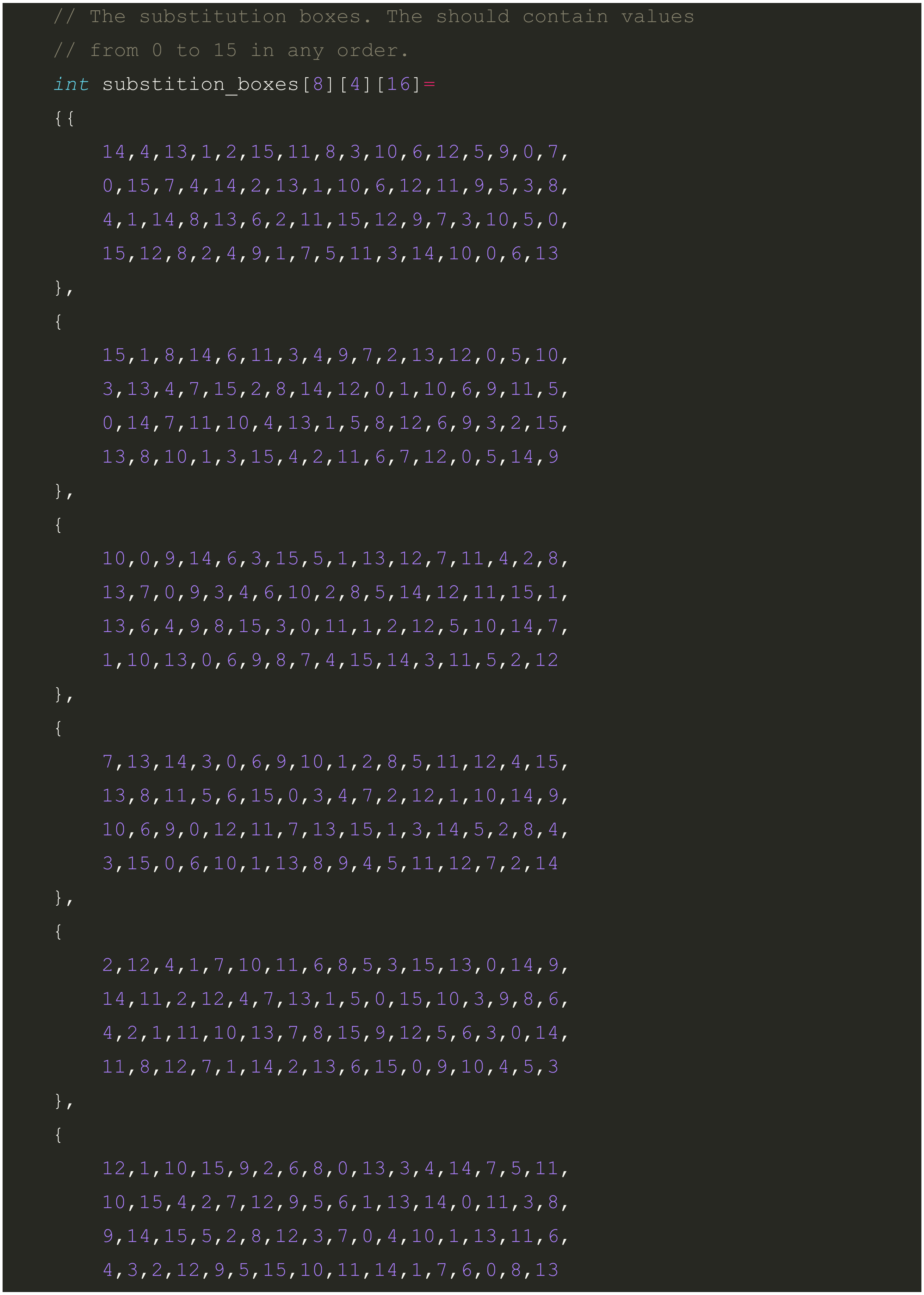
```
// Function to do a circular left shift by 1
string shift_left_once(string key_chunk){
    string shifted="";
        for(int i = 1; i < 28; i++){
            shifted += key_chunk[i];
        }
        shifted += key_chunk[0];
    return shifted;
}
// Function to do a circular left shift by 2
string shift_left_twice(string key_chunk){
    string shifted="";
    for(int i = 0; i < 2; i++){
        for(int j = 1; j < 28; j++){
            shifted += key_chunk[j];
        }
        shifted += key_chunk[0];
        key_chunk= shifted;
        shifted ="";
    }
    return key_chunk;
}
// Function to compute xor between two strings
string Xor(string a, string b){
    string result = "";
    int size = b.size();
    for(int i = 0; i < size; i++){
        if(a[i] != b[i]){
            result += "1";
        }
        else{
            result += "0";
        }
    }
    return result;
}
// Function to generate the 16 keys.
void generate_keys(string key){
```



```
// The PC1 table
int pc1[56] = {
57,49,41,33,25,17,9,
1,58,50,42,34,26,18,
10,2,59,51,43,35,27,
19,11,3,60,52,44,36,
63,55,47,39,31,23,15,
7,62,54,46,38,30,22,
14,6,61,53,45,37,29,
21,13,5,28,20,12,4
};
// The PC2 table
int pc2[48] = {
14,17,11,24,1,5,
3,28,15,6,21,10,
23,19,12,4,26,8,
16,7,27,20,13,2,
41,52,31,37,47,55,
30,40,51,45,33,48,
44,49,39,56,34,53,
46,42,50,36,29,32
};
// 1. Compressing the key using the PC1 table
string perm_key ="";
for(int i = 0; i < 56; i++){
    perm_key+= key[pc1[i]-1];
}
// 2. Dividing the key into two equal halves
string left= perm_key.substr(0, 28);
string right= perm_key.substr(28, 28);
for(int i=0; i<16; i++){
    // 3.1. For rounds 1, 2, 9, 16 the key_chunks
    // are shifted by one.
    if(i == 0 || i == 1 || i==8 || i==15 ){
        left= shift_left_once(left);
        right= shift_left_once(right);
    }
    // 3.2. For other rounds, the key_chunks
```



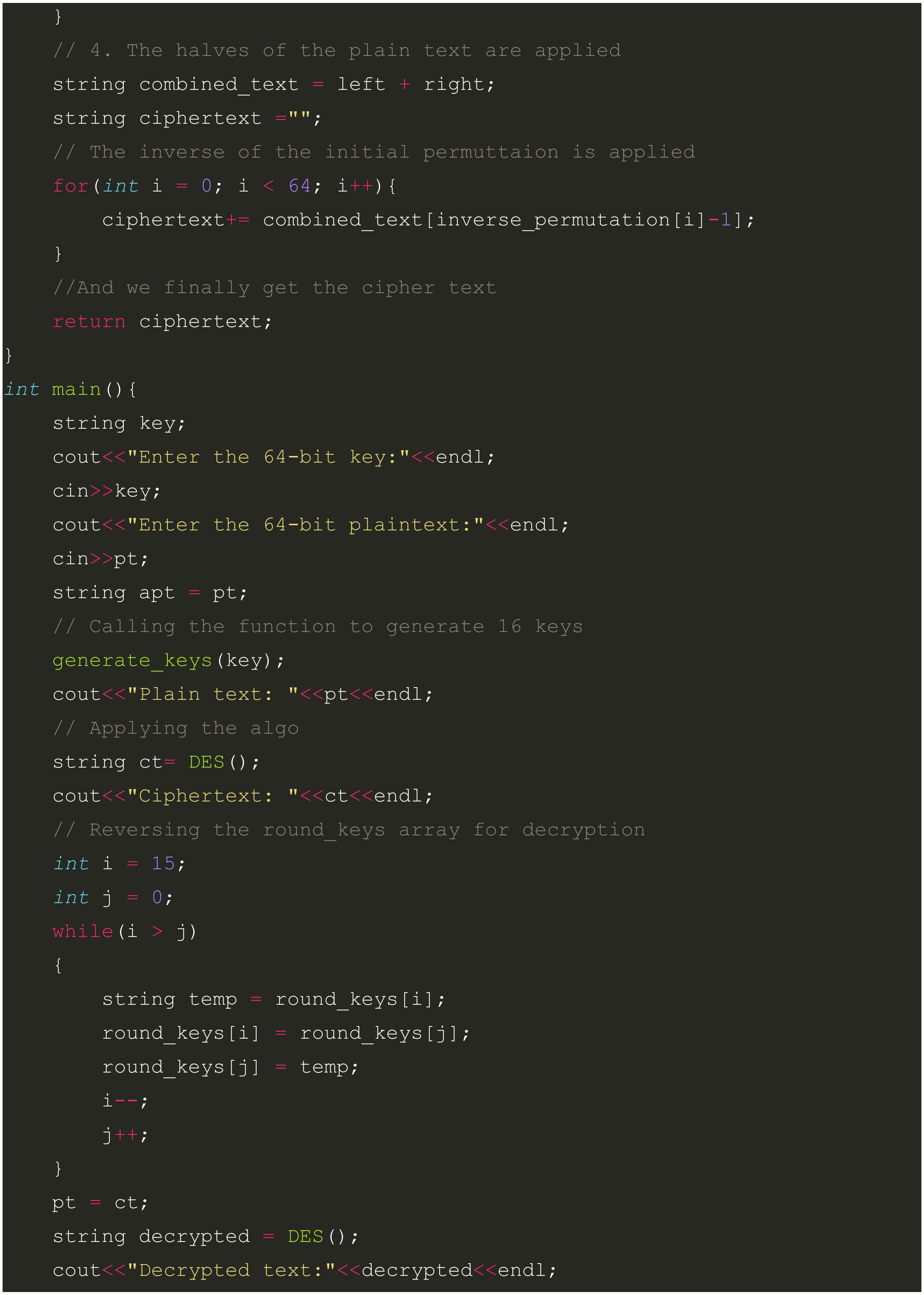
```
        // are shifted by two
        else{
            left= shift_left_twice(left);
            right= shift_left_twice(right);
        }
        // Combining the two chunks
        string combined_key = left + right;
        string round_key = "";
        // Finally, using the PC2 table to transpose the key bits
        for(int i = 0; i < 48; i++){
            round_key += combined_key[pc2[i]-1];
        }
        round_keys[i] = round_key;
    }

}
// Implementing the algorithm
string DES(){
    // The initial permutation table
    int initial_permutation[64] = {
    58,50,42,34,26,18,10,2,
    60,52,44,36,28,20,12,4,
    62,54,46,38,30,22,14,6,
    64,56,48,40,32,24,16,8,
    57,49,41,33,25,17,9,1,
    59,51,43,35,27,19,11,3,
    61,53,45,37,29,21,13,5,
    63,55,47,39,31,23,15,7
    };
    // The expansion table
    int expansion_table[48] = {
    32,1,2,3,4,5,4,5,
    6,7,8,9,8,9,10,11,
    12,13,12,13,14,15,16,17,
    16,17,18,19,20,21,20,21,
    22,23,24,25,24,25,26,27,
    28,29,28,29,30,31,32,1
    };
```



```
    // The substitution boxes. The should contain values
    // from 0 to 15 in any order.
    int substition_boxes[8][4][16]=
    {{
        14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
        0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
        4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
        15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13
    },
    {
        15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
        3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
        0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
        13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9
    },
    {
        10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,
        13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,
        13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,
        1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12
    },
    {
        7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,
        13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,
        10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,
        3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14
    },
    {
        2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
        14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
        4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
        11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3
    },
    {
        12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
        10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
        9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
        4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13
```

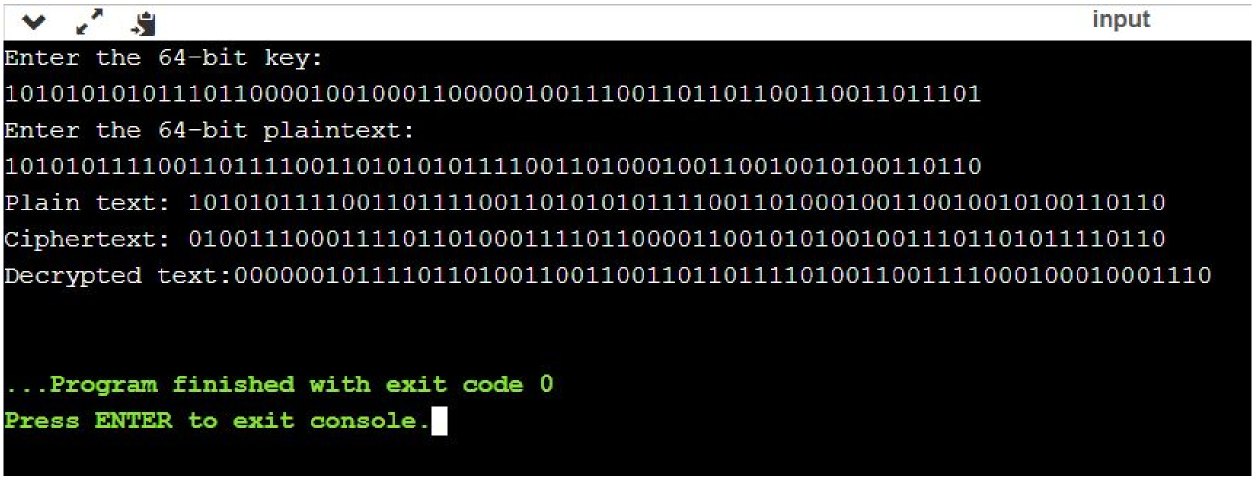
```
},
{
    4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
    13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,
    1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
    6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12
},
{
    13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
    1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
    7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
    2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11
}};
// The permutation table
int permutation_tab[32] = {
16,7,20,21,29,12,28,17,
1,15,23,26,5,18,31,10,
2,8,24,14,32,27,3,9,
19,13,30,6,22,11,4,25
};
// The inverse permutation table
int inverse_permutation[64]= {
40,8,48,16,56,24,64,32,
39,7,47,15,55,23,63,31,
38,6,46,14,54,22,62,30,
37,5,45,13,53,21,61,29,
36,4,44,12,52,20,60,28,
35,3,43,11,51,19,59,27,
34,2,42,10,50,18,58,26,
33,1,41,9,49,17,57,25
};
//1. Applying the initial permutation
string perm = "";
for(int i = 0; i < 64; i++){
    perm += pt[initial_permutation[i]-1];
}
// 2. Dividing the result into two equal halves
string left = perm.substr(0, 32);
```

```
    string right = perm.substr(32, 32);
    // The plain text is encrypted 16 times
    for(int i=0; i<16; i++) {
        string right_expanded = "";
        // 3.1. The right half of the plain text is expanded
        for(int i = 0; i < 48; i++) {
            right_expanded += right[expansion_table[i]-1];
    };  // 3.3. The result is xored with a key
        string xored = Xor(round_keys[i], right_expanded);
        string res = "";
        // 3.4. The result is divided into 8 equal parts and passed
        // through 8 substitution boxes. After passing through a
        // substituion box, each box is reduces from 6 to 4 bits.
        for(int i=0;i<8; i++){
            // Finding row and column indices to lookup the
            // substituition box
            string row1= xored.substr(i*6,1) + xored.substr(i*6 + 5,1);
            int row = convertBinaryToDecimal(row1);
            string col1 = xored.substr(i*6 + 1,1) + xored.substr(i*6 +
2,1) + xored.substr(i*6 + 3,1) + xored.substr(i*6 + 4,1);;
            int col = convertBinaryToDecimal(col1);
            int val = substition_boxes[i][row][col];
            res += convertDecimalToBinary(val);
        }
        // 3.5. Another permutation is applied
        string perm2 ="";
        for(int i = 0; i < 32; i++){
            perm2 += res[permutation_tab[i]-1];
        }
        // 3.6. The result is xored with the left half
        xored = Xor(perm2, left);
        // 3.7. The left and the right parts of the plain text are swapped
        left = xored;
        if(i < 15){
            string temp = right;
            right = xored;
            left = temp;
        }
```



```cpp
    }
    // 4. The halves of the plain text are applied
    string combined_text = left + right;
    string ciphertext ="";
    // The inverse of the initial permuttaion is applied
    for(int i = 0; i < 64; i++){
        ciphertext+= combined_text[inverse_permutation[i]-1];
    }
    //And we finally get the cipher text
    return ciphertext;
}
int main(){
    string key;
    cout<<"Enter the 64-bit key:"<<endl;
    cin>>key;
    cout<<"Enter the 64-bit plaintext:"<<endl;
    cin>>pt;
    string apt = pt;
    // Calling the function to generate 16 keys
    generate_keys(key);
    cout<<"Plain text: "<<pt<<endl;
    // Applying the algo
    string ct= DES();
    cout<<"Ciphertext: "<<ct<<endl;
    // Reversing the round_keys array for decryption
    int i = 15;
    int j = 0;
    while(i > j)
    {
        string temp = round_keys[i];
        round_keys[i] = round_keys[j];
        round_keys[j] = temp;
        i--;
        j++;
    }
    pt = ct;
    string decrypted = DES();
    cout<<"Decrypted text:"<<decrypted<<endl;
    // Comapring the initial plain text with the decrypted text
    if (decrypted == apt){
        cout<<"Plain text encrypted and decrypted successfully."<<endl;
    }
}
```

OUTPUT:



```
Enter the 64-bit key:
1010101010111011000010010001100000100111001101101100110011011101
Enter the 64-bit plaintext:
1010101111001101111001101010101111001101000100110010010100110110
Plain text: 1010101111001101111001101010101111001101000100110010010100110110
Ciphertext: 0100111000111101101000111101100001100101010010011101101011110110
Decrypted text:000000101111101101001100110011011011111010011001111000100010001110


...Program finished with exit code 0
Press ENTER to exit console.
```

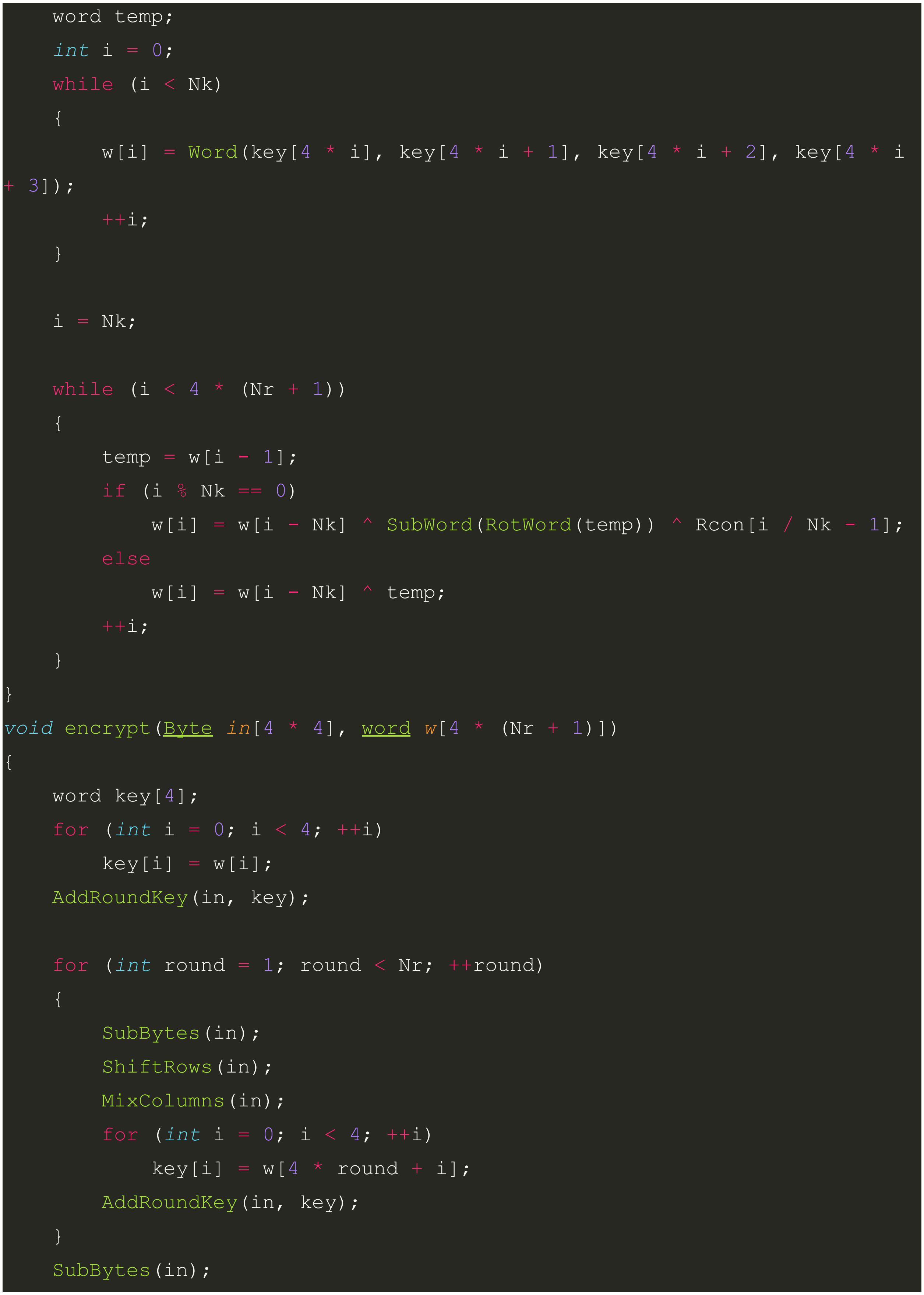
AES

Code:

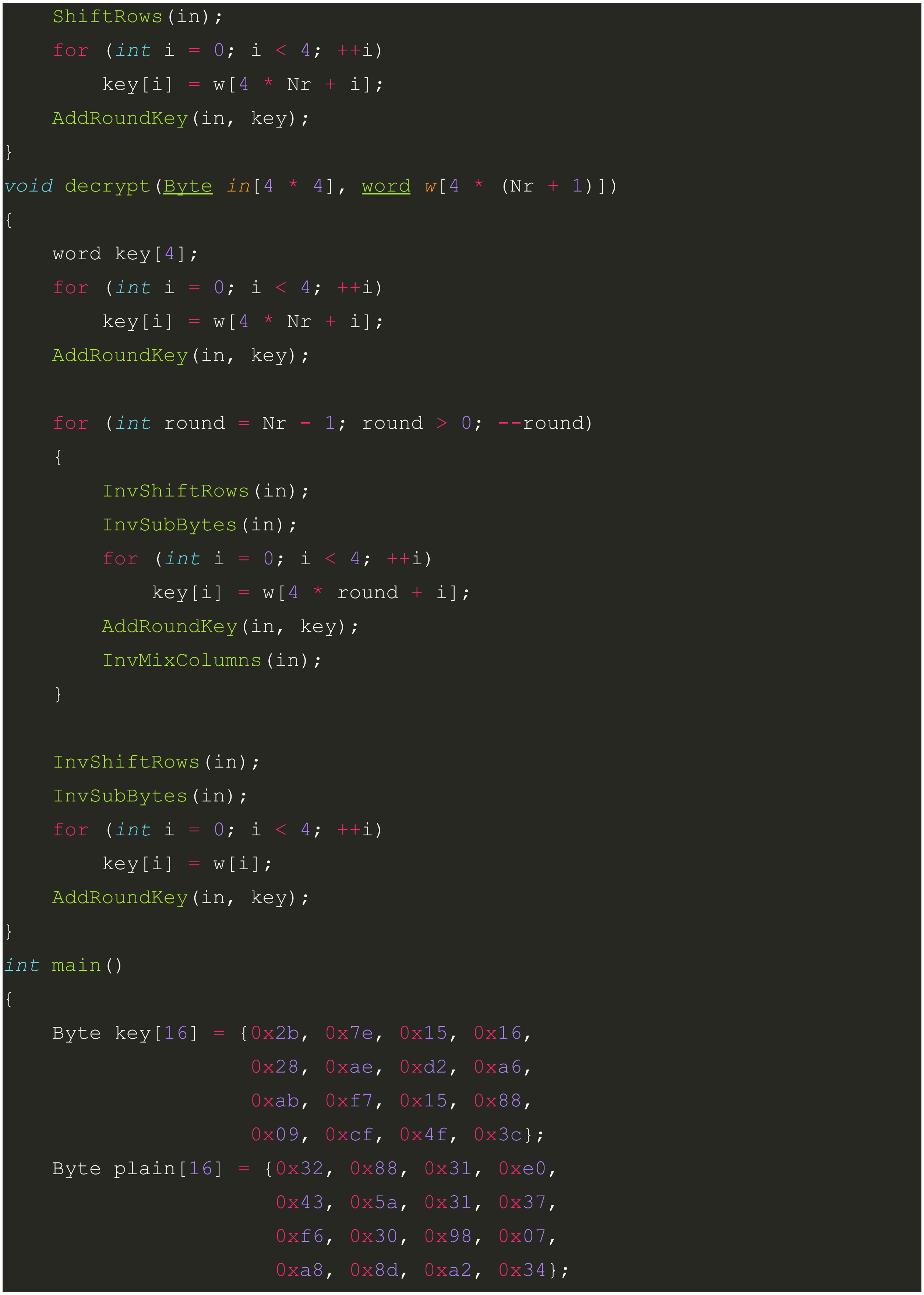
```
#include <iostream>
#include <bitset>
#include <string>
using namespace std;
typedef bitset<8> Byte;
typedef bitset<32> word;

const int Nr = 10;
const int Nk = 4;

Byte S_Box[16][16] = {
    {0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67,
0x2B, 0xFE, 0xD7, 0xAB, 0x76},
    {0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2,
0xAF, 0x9C, 0xA4, 0x72, 0xC0},
    {0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5,
0xF1, 0x71, 0xD8, 0x31, 0x15},
    {0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80,
0xE2, 0xEB, 0x27, 0xB2, 0x75},
    {0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6,
0xB3, 0x29, 0xE3, 0x2F, 0x84},
    {0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE,
0x39, 0x4A, 0x4C, 0x58, 0xCF},
    {0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02,
0x7F, 0x50, 0x3C, 0x9F, 0xA8},
    {0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA,
0x21, 0x10, 0xFF, 0xF3, 0xD2},
    {0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E,
0x3D, 0x64, 0x5D, 0x19, 0x73},
    {0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8,
0x14, 0xDE, 0x5E, 0x0B, 0xDB},
    {0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC,
0x62, 0x91, 0x95, 0xE4, 0x79},
    {0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4,
0xEA, 0x65, 0x7A, 0xAE, 0x08},
    {0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A},
    {0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E},
    {0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF},
    {0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16}};

Byte Inv_S_Box[16][16] = {
    {0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB},
    {0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB},
    {0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E},
    {0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25},
    {0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92},
    {0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84},
    {0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06},
    {0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B},
    {0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73},
    {0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E},
    {0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B},
    {0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4},
    {0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F},
    {0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF},
    {0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB,
0x3C, 0x83, 0x53, 0x99, 0x61},
    {0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14,
0x63, 0x55, 0x21, 0x0C, 0x7D}};

word Rcon[10] = {0x01000000, 0x02000000, 0x04000000, 0x08000000,
0x10000000,
                 0x20000000, 0x40000000, 0x80000000, 0x1b000000,
0x36000000};

void SubBytes(Byte mtx[4 * 4])
{
    for (int i = 0; i < 16; ++i)
    {
        int row = mtx[i][7] * 8 + mtx[i][6] * 4 + mtx[i][5] * 2 +
mtx[i][4];
        int col = mtx[i][3] * 8 + mtx[i][2] * 4 + mtx[i][1] * 2 +
mtx[i][0];
        mtx[i] = S_Box[row][col];
    }
}

void ShiftRows(Byte mtx[4 * 4])
{
    Byte temp = mtx[4];
    for (int i = 0; i < 3; ++i)
        mtx[i + 4] = mtx[i + 5];
    mtx[7] = temp;
    for (int i = 0; i < 2; ++i)
    {
        temp = mtx[i + 8];
        mtx[i + 8] = mtx[i + 10];
        mtx[i + 10] = temp;
    }
    temp = mtx[15];
    for (int i = 3; i > 0; --i)
        mtx[i + 12] = mtx[i + 11];
    mtx[12] = temp;
```

```
}

Byte GFMul(Byte a, Byte b)
{
    Byte p = 0;
    Byte hi_bit_set;
    for (int counter = 0; counter < 8; counter++)
    {
        if ((b & Byte(1)) != 0)
        {
            p ^= a;
        }
        hi_bit_set = (Byte)(a & Byte(0x80));
        a <<= 1;
        if (hi_bit_set != 0)
        {
            a ^= 0x1b;
        }
        b >>= 1;
    }
    return p;
}

void MixColumns(Byte mtx[4 * 4])
{
    Byte arr[4];
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
            arr[j] = mtx[i + j * 4];

        mtx[i] = GFMul(0x02, arr[0]) ^ GFMul(0x03, arr[1]) ^ arr[2] ^
arr[3];
        mtx[i + 4] = arr[0] ^ GFMul(0x02, arr[1]) ^ GFMul(0x03, arr[2]) ^
arr[3];
        mtx[i + 8] = arr[0] ^ arr[1] ^ GFMul(0x02, arr[2]) ^ GFMul(0x03,
arr[3]);
```

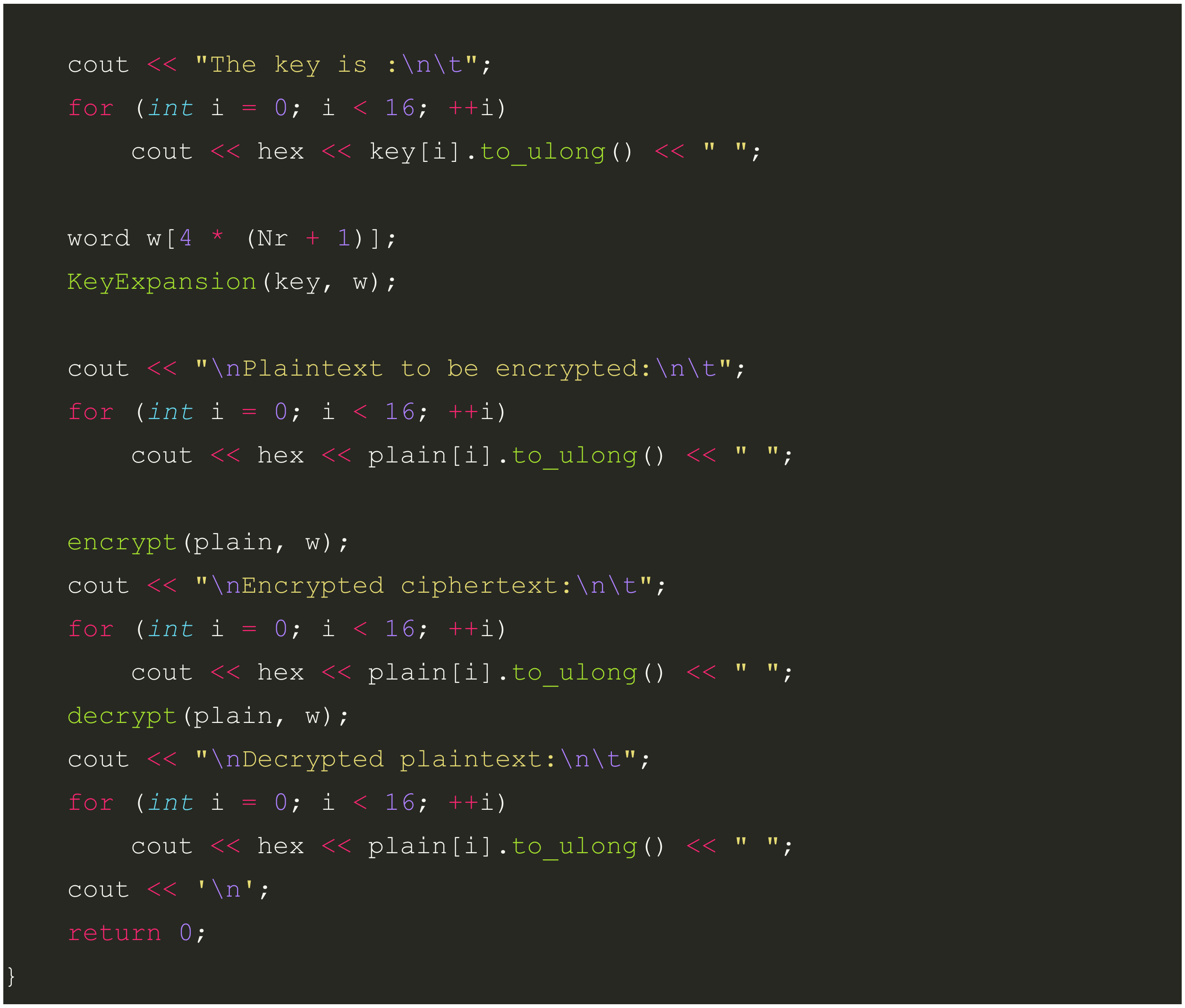
```
        mtx[i + 12] = GFMul(0x03, arr[0]) ^ arr[1] ^ arr[2] ^ GFMul(0x02,
arr[3]);
    }
}

void AddRoundKey(Byte mtx[4 * 4], word k[4])
{
    for (int i = 0; i < 4; ++i)
    {
        word k1 = k[i] >> 24;
        word k2 = (k[i] << 8) >> 24;
        word k3 = (k[i] << 16) >> 24;
        word k4 = (k[i] << 24) >> 24;

        mtx[i] = mtx[i] ^ Byte(k1.to_ulong());
        mtx[i + 4] = mtx[i + 4] ^ Byte(k2.to_ulong());
        mtx[i + 8] = mtx[i + 8] ^ Byte(k3.to_ulong());
        mtx[i + 12] = mtx[i + 12] ^ Byte(k4.to_ulong());
    }
}

void InvSubBytes(Byte mtx[4 * 4])
{
    for (int i = 0; i < 16; ++i)
    {
        int row = mtx[i][7] * 8 + mtx[i][6] * 4 + mtx[i][5] * 2 +
mtx[i][4];
        int col = mtx[i][3] * 8 + mtx[i][2] * 4 + mtx[i][1] * 2 +
mtx[i][0];
        mtx[i] = Inv_S_Box[row][col];
    }
}

void InvShiftRows(Byte mtx[4 * 4])
{
    Byte temp = mtx[7];
    for (int i = 3; i > 0; --i)
        mtx[i + 4] = mtx[i + 3];
    mtx[4] = temp;

    for (int i = 0; i < 2; ++i)
    {
        temp = mtx[i + 8];
        mtx[i + 8] = mtx[i + 10];
        mtx[i + 10] = temp;
    }

    temp = mtx[12];
    for (int i = 0; i < 3; ++i)
        mtx[i + 12] = mtx[i + 13];
    mtx[15] = temp;
}

void InvMixColumns(Byte mtx[4 * 4])
{
    Byte arr[4];
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
            arr[j] = mtx[i + j * 4];

        mtx[i] = GFMul(0x0e, arr[0]) ^ GFMul(0x0b, arr[1]) ^ GFMul(0x0d, arr[2]) ^ GFMul(0x09, arr[3]);
        mtx[i + 4] = GFMul(0x09, arr[0]) ^ GFMul(0x0e, arr[1]) ^ GFMul(0x0b, arr[2]) ^ GFMul(0x0d, arr[3]);
        mtx[i + 8] = GFMul(0x0d, arr[0]) ^ GFMul(0x09, arr[1]) ^ GFMul(0x0e, arr[2]) ^ GFMul(0x0b, arr[3]);
        mtx[i + 12] = GFMul(0x0b, arr[0]) ^ GFMul(0x0d, arr[1]) ^ GFMul(0x09, arr[2]) ^ GFMul(0x0e, arr[3]);
    }
}

word Word(Byte &k1, Byte &k2, Byte &k3, Byte &k4)
{
    word result(0x00000000);
    word temp;
    temp = k1.to_ulong();
    temp <<= 24;
    result |= temp;
    temp = k2.to_ulong();
    temp <<= 16;
    result |= temp;
    temp = k3.to_ulong();
    temp <<= 8;
    result |= temp;
    temp = k4.to_ulong();
    result |= temp;
    return result;
}

word RotWord(word rw)
{
    word high = rw << 8;
    word low = rw >> 24;
    return high | low;
}

word SubWord(word sw)
{
    word temp;
    for (int i = 0; i < 32; i += 8)
    {
        int row = sw[i + 7] * 8 + sw[i + 6] * 4 + sw[i + 5] * 2 + sw[i +
4];
        int col = sw[i + 3] * 8 + sw[i + 2] * 4 + sw[i + 1] * 2 + sw[i];
        Byte val = S_Box[row][col];
        for (int j = 0; j < 8; ++j)
            temp[i + j] = val[j];
    }
    return temp;
}

void KeyExpansion(Byte key[4 * Nk], word w[4 * (Nr + 1)])
{
```



```
    word temp;
    int i = 0;
    while (i < Nk)
    {
        w[i] = Word(key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i
+ 3]);
        ++i;
    }

    i = Nk;

    while (i < 4 * (Nr + 1))
    {
        temp = w[i - 1];
        if (i % Nk == 0)
            w[i] = w[i - Nk] ^ SubWord(RotWord(temp)) ^ Rcon[i / Nk - 1];
        else
            w[i] = w[i - Nk] ^ temp;
        ++i;
    }
}
void encrypt(Byte in[4 * 4], word w[4 * (Nr + 1)])
{
    word key[4];
    for (int i = 0; i < 4; ++i)
        key[i] = w[i];
    AddRoundKey(in, key);

    for (int round = 1; round < Nr; ++round)
    {
        SubBytes(in);
        ShiftRows(in);
        MixColumns(in);
        for (int i = 0; i < 4; ++i)
            key[i] = w[4 * round + i];
        AddRoundKey(in, key);
    }
    SubBytes(in);
```

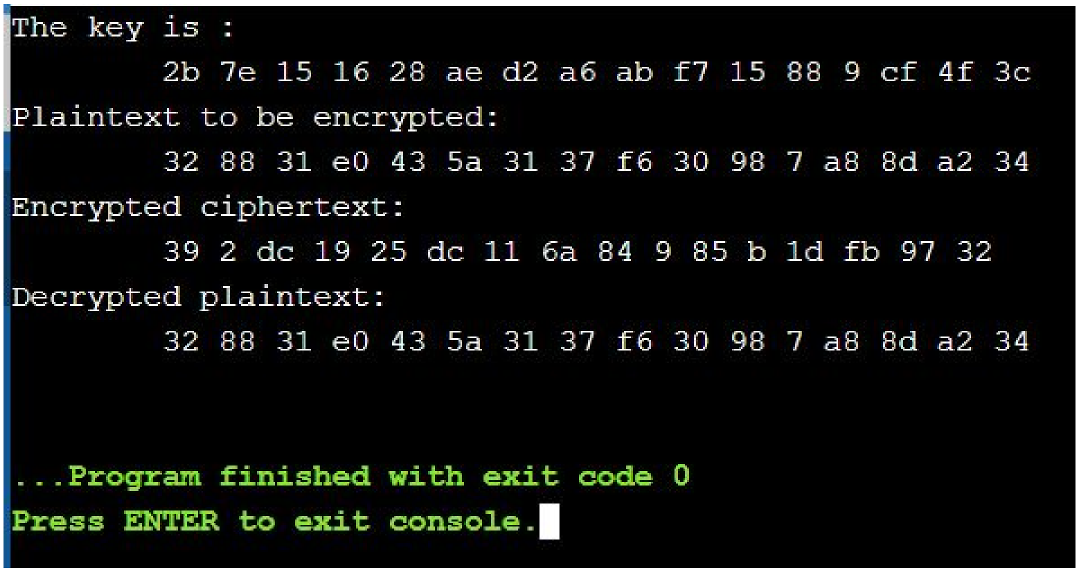


```
    ShiftRows(in);
    for (int i = 0; i < 4; ++i)
        key[i] = w[4 * Nr + i];
    AddRoundKey(in, key);
}
void decrypt(Byte in[4 * 4], word w[4 * (Nr + 1)])
{
    word key[4];
    for (int i = 0; i < 4; ++i)
        key[i] = w[4 * Nr + i];
    AddRoundKey(in, key);

    for (int round = Nr - 1; round > 0; --round)
    {
        InvShiftRows(in);
        InvSubBytes(in);
        for (int i = 0; i < 4; ++i)
            key[i] = w[4 * round + i];
        AddRoundKey(in, key);
        InvMixColumns(in);
    }

    InvShiftRows(in);
    InvSubBytes(in);
    for (int i = 0; i < 4; ++i)
        key[i] = w[i];
    AddRoundKey(in, key);
}
int main()
{
    Byte key[16] = {0x2b, 0x7e, 0x15, 0x16,
                    0x28, 0xae, 0xd2, 0xa6,
                    0xab, 0xf7, 0x15, 0x88,
                    0x09, 0xcf, 0x4f, 0x3c};
    Byte plain[16] = {0x32, 0x88, 0x31, 0xe0,
                      0x43, 0x5a, 0x31, 0x37,
                      0xf6, 0x30, 0x98, 0x07,
                      0xa8, 0x8d, 0xa2, 0x34};
```



```
    cout << "The key is :\n\t";
    for (int i = 0; i < 16; ++i)
        cout << hex << key[i].to_ulong() << " ";

    word w[4 * (Nr + 1)];
    KeyExpansion(key, w);

    cout << "\nPlaintext to be encrypted:\n\t";
    for (int i = 0; i < 16; ++i)
        cout << hex << plain[i].to_ulong() << " ";

    encrypt(plain, w);
    cout << "\nEncrypted ciphertext:\n\t";
    for (int i = 0; i < 16; ++i)
        cout << hex << plain[i].to_ulong() << " ";
    decrypt(plain, w);
    cout << "\nDecrypted plaintext:\n\t";
    for (int i = 0; i < 16; ++i)
        cout << hex << plain[i].to_ulong() << " ";
    cout << '\n';
    return 0;
}
```

OUtput:



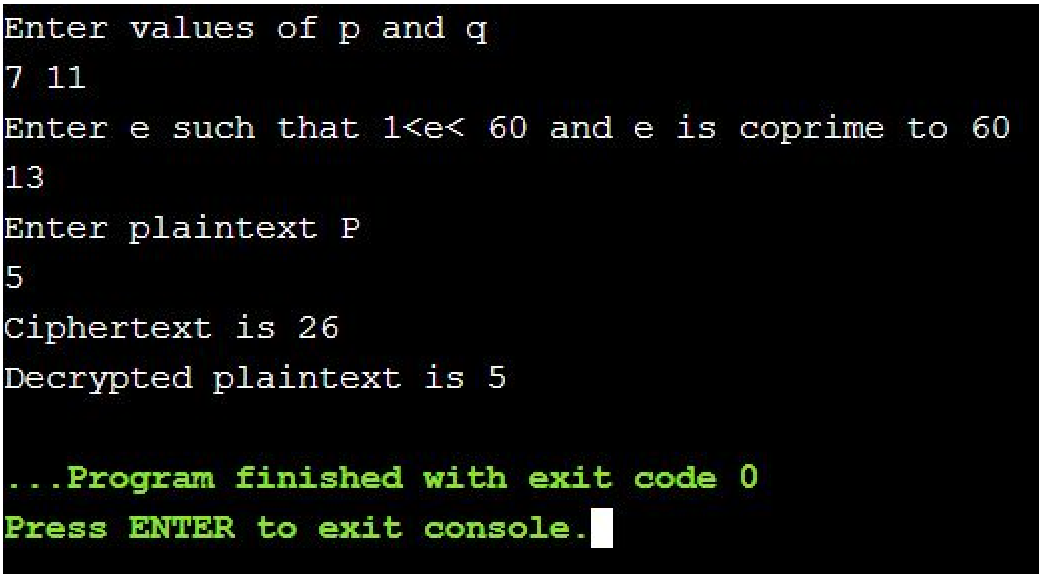
```
The key is :
        2b 7e 15 16 28 ae d2 a6 ab f7 15 88 9 cf 4f 3c
Plaintext to be encrypted:
        32 88 31 e0 43 5a 31 37 f6 30 98 7 a8 8d a2 34
Encrypted ciphertext:
        39 2 dc 19 25 dc 11 6a 84 9 85 b 1d fb 97 32
Decrypted plaintext:
        32 88 31 e0 43 5a 31 37 f6 30 98 7 a8 8d a2 34


...Program finished with exit code 0
Press ENTER to exit console.
```

RSA

Code:

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

int inverse(int r, int n){
    r = r%n;
    for (int x=1; x<n; x++)
       if ((r*x) % n == 1)
          return x;
}

int fastExponentiation(int a, int x, int n){
    int y = 1;     // Initialize result
    while (x > 0){
        // If y is odd, multiply x with result
        if (x & 1)
            y = (a*y) % n;
        x = x>>1;
        a = (a*a) % n;
    }
    return y;
}
int main(){
    //Key Generation
    int p, q;
    cout<<"Enter values of p and q \n";
    cin>>p>>q;
    int n=p*q;
    int phi_n=(p-1)*(q-1);
    cout<<"Enter e such that 1<e< "<<phi_n<<" and e is coprime to
"<<phi_n<<'\n';
    int e;
    cin>>e;
    int d=inverse(e, phi_n);

    //Encryption
    int P;
    cout<<"Enter plaintext P \n";
    cin>>P;
    int C=fastExponentiation(P, e, n);
    cout<<"Ciphertext is "<<C<<'\n';

    //Decryption
    int decrypted=fastExponentiation(C, d, n);
    cout<<"Decrypted plaintext is "<<decrypted;
}
```

OUTPUT:

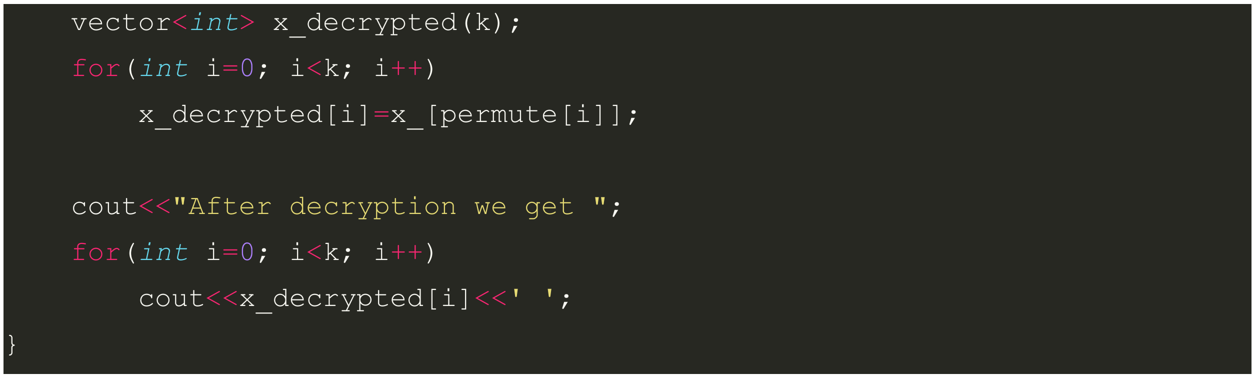


```
Enter values of p and q
7 11
Enter e such that 1<e< 60 and e is coprime to 60
13
Enter plaintext P
5
Ciphertext is 26
Decrypted plaintext is 5

...Program finished with exit code 0
Press ENTER to exit console.
```

KNAPSACK CRYPTOSYSTEM

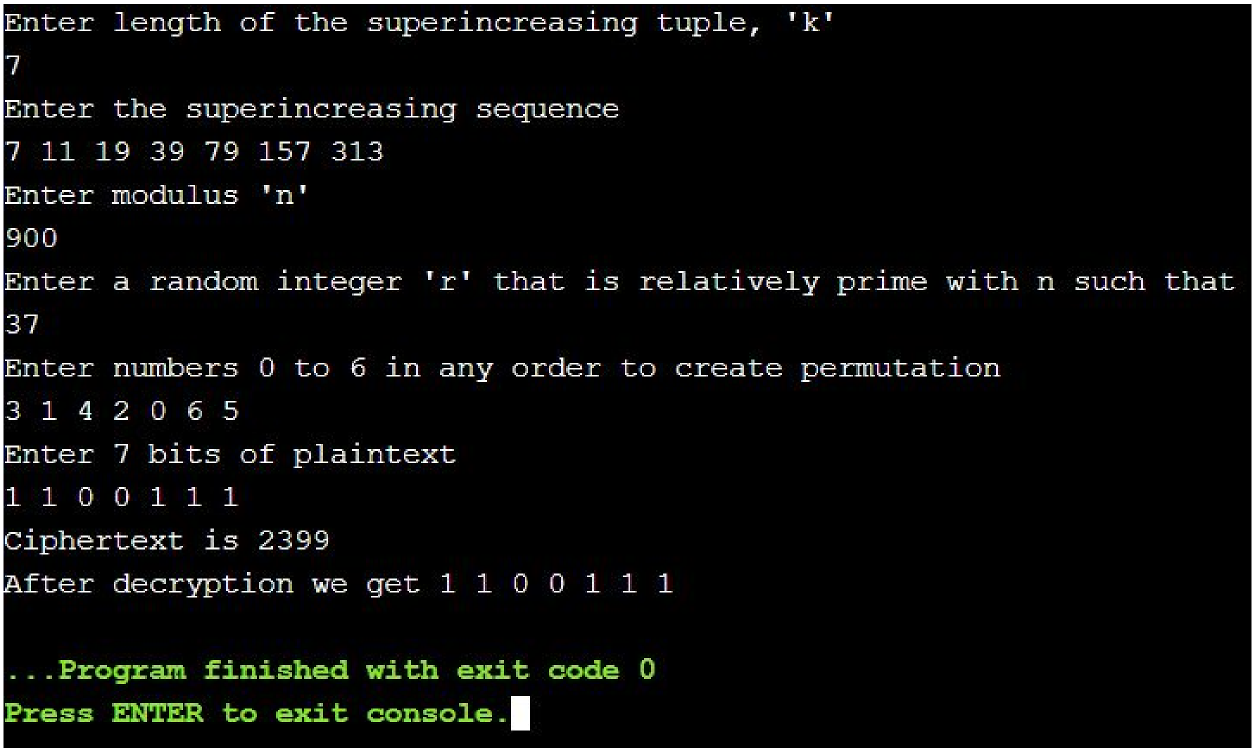
Code:

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

int inverse(int r, int n){
    r = r%n;
    for (int x=1; x<n; x++)
       if ((r*x) % n == 1)
          return x;
}

int knapsackSum(vector<int> x, vector<int> a){
    int sum=0;
    for(int i=0; i<x.size(); i++)
        sum+=a[i]*x[i];
    return sum;
}

vector<int> inv_knapsackSum(int s, vector<int> a){
    vector<int> x(a.size());
    for(int i=a.size()-1; i>=0; i--)
        if(s>=a[i]){
            x[i]=1;
            s-=a[i];
        }
    return x;
}

int main(){
    //Key generation
    cout<<"Enter length of the superincreasing tuple, 'k' \n";
    int k;
    cin>>k;

    cout<<"Enter the superincreasing sequence \n";
    vector<int> b(k);
    for(int i=0; i<k; i++)
        cin>>b[i];

    cout<<"Enter modulus 'n' \n";
    int n;
    cin>>n;

    cout<<"Enter a random integer 'r' that is relatively prime with n such
that 1<=r<n \n";
    int r;
    cin>>r;

    vector<int> t(k);
    for(int i=0; i<k; i++)
        t[i]=r*b[i]%n;

    vector<int> permute(k);
    cout<<"Enter numbers 0 to "<<k-1<<" in any order to create permutation
\n";
    for(int i=0; i<k; i++)
        cin>>permute[i];

    vector<int> a(k);
    for(int i=0; i<k; i++)
        a[i]=t[permute[i]];

    //Encryption
    cout<<"Enter "<<k<<" bits of plaintext \n";
    vector<int> x(k);
    for(int i=0; i<k; i++)
        cin>>x[i];

    int s=knapsackSum(a, x);
    cout<<"Ciphertext is "<<s<<'\n';

    //Decryption
    int s_=s*inverse(r,n)%n;
    vector<int> x_=inv_knapsackSum(s_, b);
```



```
    vector<int> x_decrypted(k);
    for(int i=0; i<k; i++)
        x_decrypted[i]=x_[permute[i]];

    cout<<"After decryption we get ";
    for(int i=0; i<k; i++)
        cout<<x_decrypted[i]<<' ';
}
```

OUTPUT:

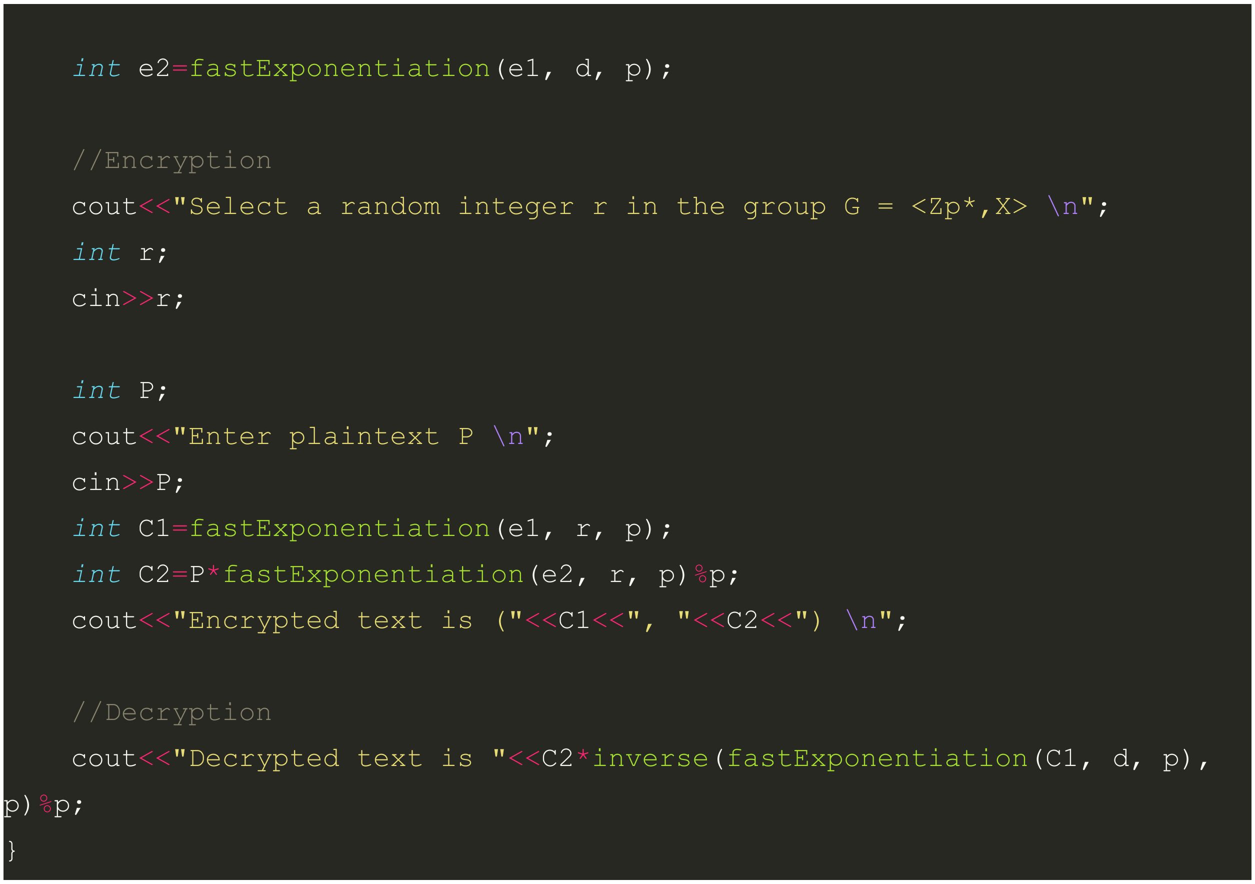


```
Enter length of the superincreasing tuple, 'k'
7
Enter the superincreasing sequence
7 11 19 39 79 157 313
Enter modulus 'n'
900
Enter a random integer 'r' that is relatively prime with n such that
37
Enter numbers 0 to 6 in any order to create permutation
3 1 4 2 0 6 5
Enter 7 bits of plaintext
1 1 0 0 1 1 1
Ciphertext is 2399
After decryption we get 1 1 0 0 1 1 1

...Program finished with exit code 0
Press ENTER to exit console.
```

ELGAMAL CRYPTOSYSTEM

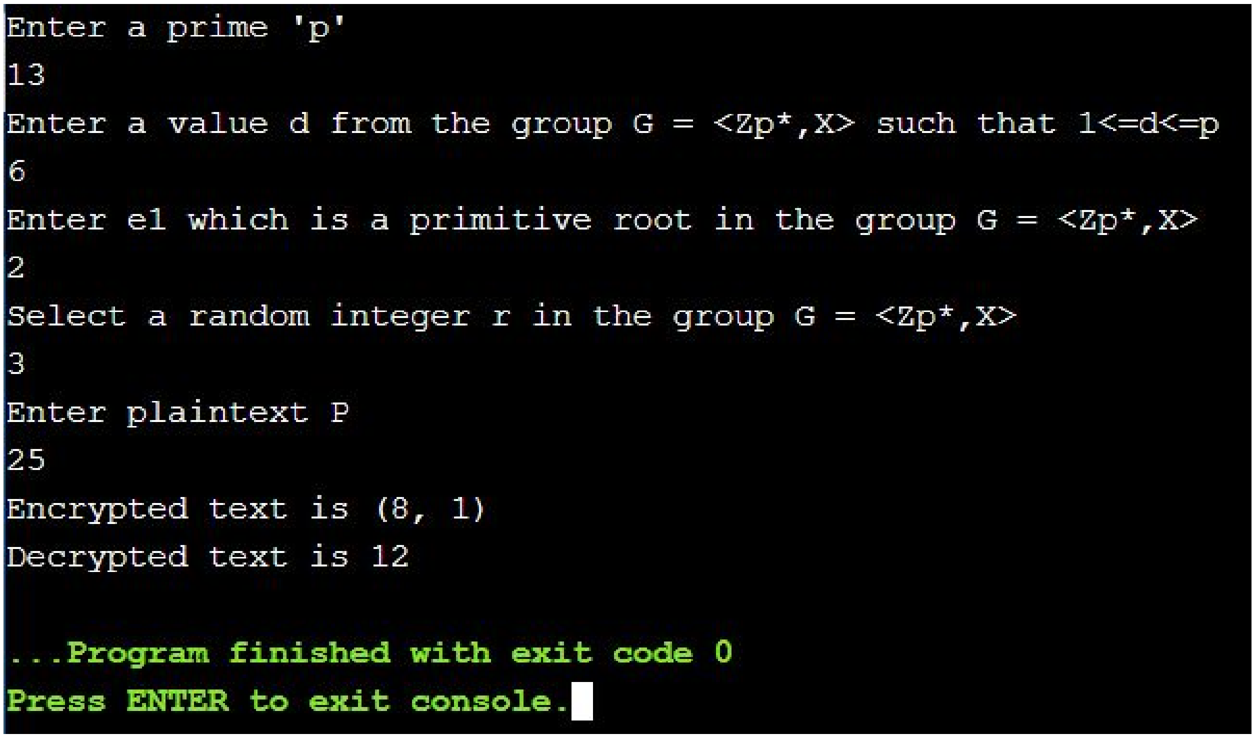
Code:

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

int inverse(int r, int n){
    r = r%n;
    for (int x=1; x<n; x++)
       if ((r*x) % n == 1)
          return x;
}

int fastExponentiation(int a, int x, int n){
    int y = 1;     // Initialize result
    while (x > 0){
        // If y is odd, multiply x with result
        if (x & 1)
            y = (a*y) % n;
        x = x>>1;
        a = (a*a) % n;
    }
    return y;
}

int main(){
    //Key Generation
    int p;
    cout<<"Enter a prime 'p' \n";
    cin>>p;

    int d;
    cout<<"Enter a value d from the group G = <Zp*,X> such that 1<=d<=p \n";
    cin>>d;
    int e1;
    cout<<"Enter e1 which is a primitive root in the group G = <Zp*,X> \n";
    cin>>e1;
```



```
    int e2=fastExponentiation(e1, d, p);

    //Encryption
    cout<<"Select a random integer r in the group G = <Zp*,X> \n";
    int r;
    cin>>r;

    int P;
    cout<<"Enter plaintext P \n";
    cin>>P;
    int C1=fastExponentiation(e1, r, p);
    int C2=P*fastExponentiation(e2, r, p)%p;
    cout<<"Encrypted text is ("<<C1<<", "<<C2<<") \n";

    //Decryption
    cout<<"Decrypted text is "<<C2*inverse(fastExponentiation(C1, d, p),
p)%p;
}
```

OUTPUT:

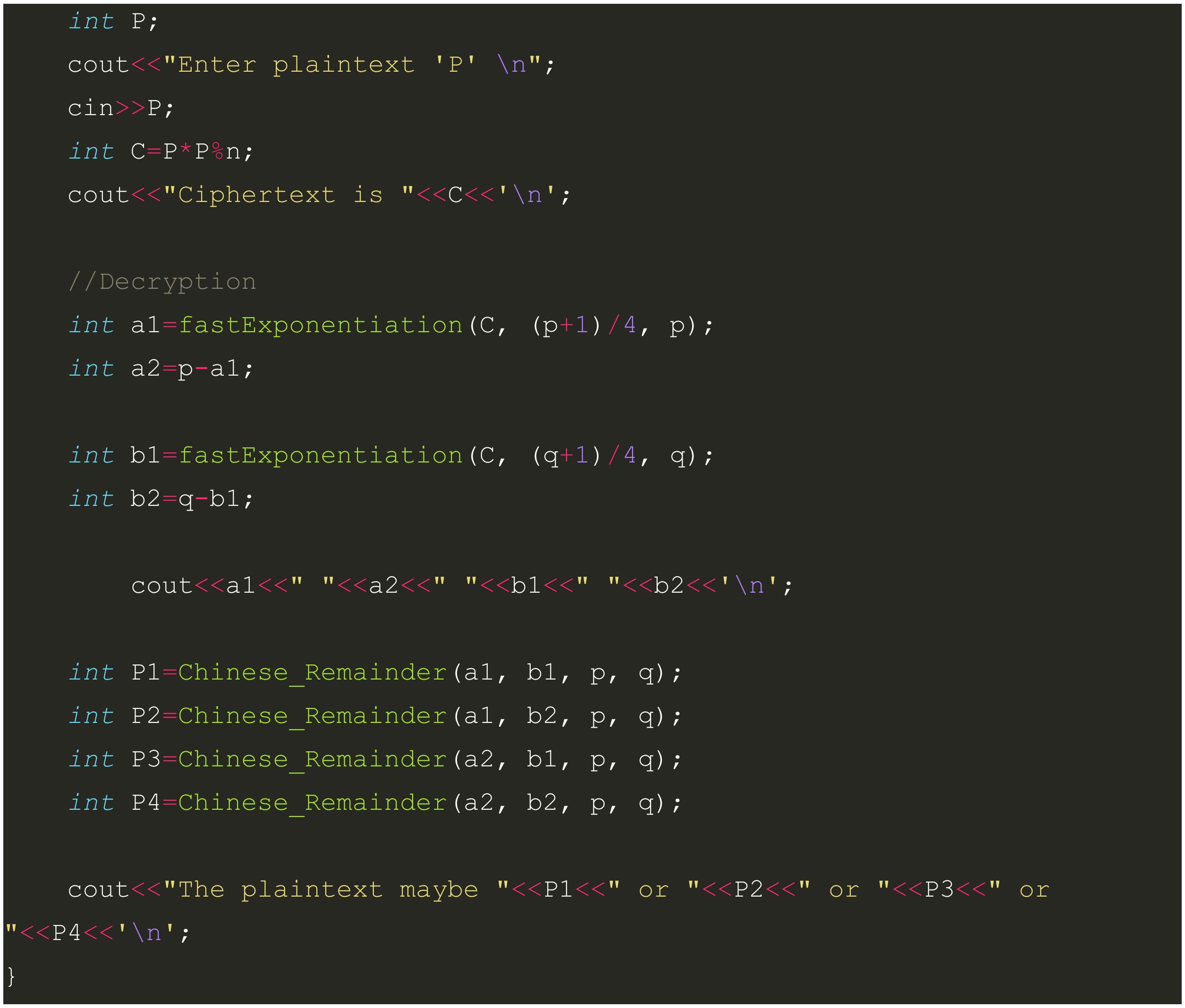


```
Enter a prime 'p'
13
Enter a value d from the group G = <Zp*,X> such that 1<=d<=p
6
Enter e1 which is a primitive root in the group G = <Zp*,X>
2
Select a random integer r in the group G = <Zp*,X>
3
Enter plaintext P
25
Encrypted text is (8, 1)
Decrypted text is 12

...Program finished with exit code 0
Press ENTER to exit console.
```

RABIN CRYPTOSYSTEM

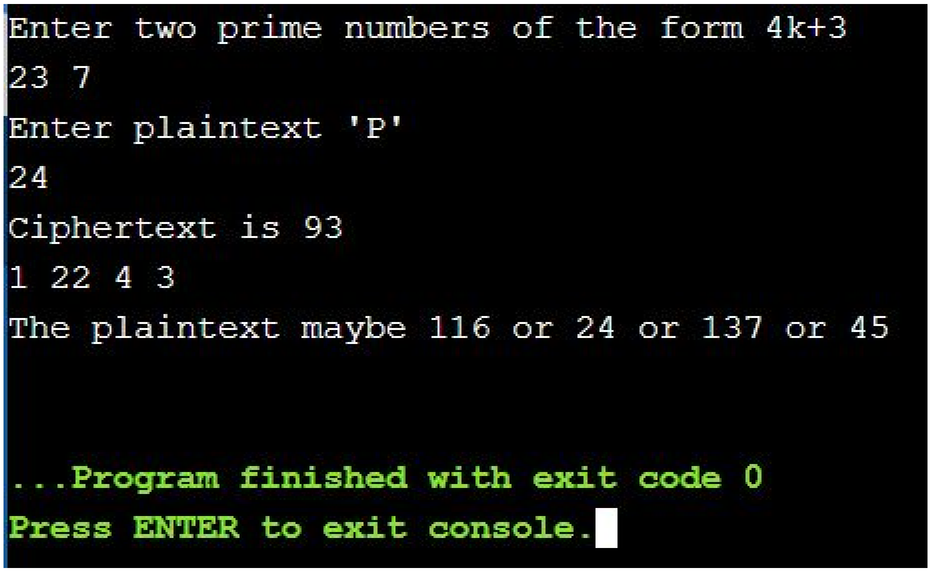
Code:

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

int inverse(int r, int n){
    r = r%n;
    for (int x=1; x<n; x++)
       if ((r*x) % n == 1)
          return x;
}

int fastExponentiation(int a, int x, int n){
    int y = 1;      // Initialize result
    while (x > 0){
        // If y is odd, multiply x with result
        if (x & 1)
            y = (a*y) % n;
        x = x>>1;
        a = (a*a) % n;
    }
    return y;
}

int Chinese_Remainder(int a1, int a2, int m1, int m2){
    int M1=inverse(m2, m1);
    int M2=inverse(m1, m2);
    return (a1*m2*M1+a2*m1*M2)%(m1*m2);
}

int main(){
    //Key Generation
    int p, q;
    cout<<"Enter two prime numbers of the form 4k+3 \n";
    cin>>p>>q;
    int n=p*q;

    //Encryption
```



```
    int P;
    cout<<"Enter plaintext 'P' \n";
    cin>>P;
    int C=P*P%n;
    cout<<"Ciphertext is "<<C<<'\n';

    //Decryption
    int a1=fastExponentiation(C, (p+1)/4, p);
    int a2=p-a1;

    int b1=fastExponentiation(C, (q+1)/4, q);
    int b2=q-b1;

        cout<<a1<<" "<<a2<<" "<<b1<<" "<<b2<<'\n';

    int P1=Chinese_Remainder(a1, b1, p, q);
    int P2=Chinese_Remainder(a1, b2, p, q);
    int P3=Chinese_Remainder(a2, b1, p, q);
    int P4=Chinese_Remainder(a2, b2, p, q);

    cout<<"The plaintext maybe "<<P1<<" or "<<P2<<" or "<<P3<<" or 
"<<P4<<'\n';
}
```

OUTPUT:



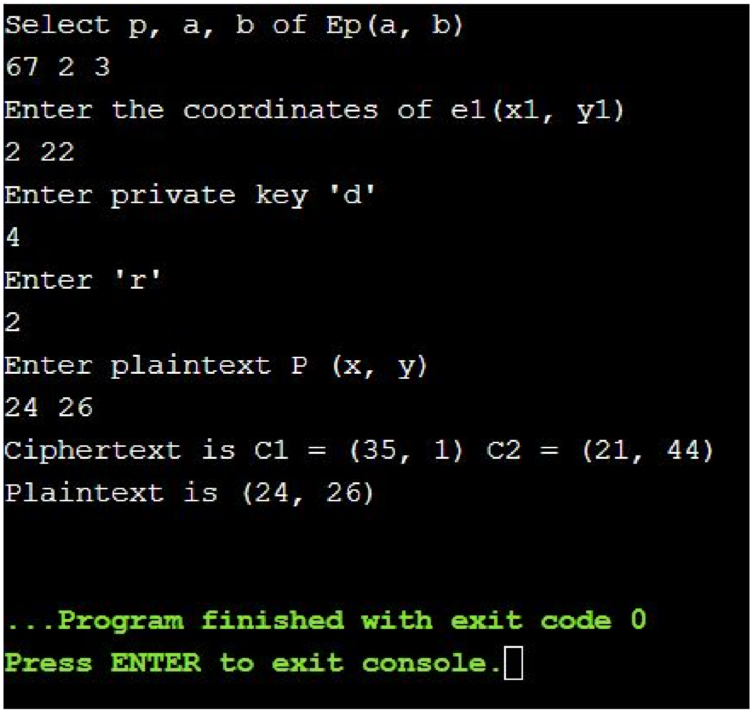
```
Enter two prime numbers of the form 4k+3
23 7
Enter plaintext 'P'
24
Ciphertext is 93
1 22 4 3
The plaintext maybe 116 or 24 or 137 or 45


...Program finished with exit code 0
Press ENTER to exit console.
```

ELLIPTIC CURVE CRYPTOSYSTEM

Code:

```
#include<iostream>
using namespace std;
int inverse(int r, int n){
    r = r%n;
    for (int x=1; x<n; x++)
       if ((r*x) % n == 1)
          return x;
}
pair<int, int> invertPoint(pair<int, int> a, int p){
    return {a.first, p-a.second};
}
pair<int, int> addPoints(pair<int, int> a, pair<int, int> b, int p, int
a1){
    int lambda;
    if(a!=b)
        lambda = (b.second+(p-a.second))*inverse(b.first+(p-a.first),
p)%p;
    else lambda = (3*a.first*a.first+a1)*inverse(2*a.second, p)%p;
    int x = (lambda*lambda%p+p-a.first+p-b.first)%p;
    int y = (lambda*(a.first+p-x)+p-a.second)%p;
    return {x, y};
}
pair<int, int> multiplyPoint(int d, pair<int, int> e, int p, int a){
    pair<int, int> result=e;
    for(int i=1; i<d; i++)
        result=addPoints(result, e, p, a);
    return result;
}
int main(){
    //Key generation
    int p, a, b, d, r;
    cout<<"Select p, a, b of Ep(a, b) \n";
    cin>>p>>a>>b;
    pair<int, int> e1;
    cout<<"Enter the coordinates of e1(x1, y1) \n";
    cin>>e1.first>>e1.second;
    cout<<"Enter private key 'd' \n";
    cin>>d;
    pair<int, int> e2=multiplyPoint(d, e1, p, a);
    //Encryption
    pair<int, int> P;
    cout<<"Enter 'r' \n";
    cin>>r;
    cout<<"Enter plaintext P (x, y) \n";
    cin>>P.first>>P.second;
    pair<int, int> C1=multiplyPoint(r, e1, p, a);
    pair<int, int> C2=addPoints(P, multiplyPoint(r, e2, p, a), p, a);
    cout<<"Ciphertext is C1 = ("<<C1.first<<", "<<C1.second<<") C2 = ("<<C2.first<<", "<<C2.second<<") \n";
    //Decryption
    pair<int, int> decrypted=addPoints(C2, invertPoint(multiplyPoint(d, C1, p, a), p), p, a);
    cout<<"Plaintext is ("<<decrypted.first<<", "<<decrypted.second<<") \n";
}
```

OUTPUT:



```
Select p, a, b of Ep(a, b)
67 2 3
Enter the coordinates of e1(x1, y1)
2 22
Enter private key 'd'
4
Enter 'r'
2
Enter plaintext P (x, y)
24 26
Ciphertext is C1 = (35, 1) C2 = (21, 44)
Plaintext is (24, 26)


...Program finished with exit code 0
Press ENTER to exit console.
```