Assignment – 9

Q1. Write a program to implement the DES algorithm.

A:

Pseudocode and Explanation –

1.Initial Permutation:

- The input 64-bit plaintext undergoes an initial permutation, rearranging bits based on a predefined table.

2. Key Scheduling:

- A 64-bit key is entered and transformed using PC-1 (permutation choice 1).
- The key is rotated in each round, and PC-2 (permutation choice 2) is applied to generate 16 subkeys, each used for one round of the Fiestel process.

3. Fiestel Cipher (16 Rounds):

- The plaintext is split into left and right halves.
- In each round:
    - The right half undergoes expansion to 48 bits.
    - It's XORed with the round key.
    - The result is passed through S-boxes for substitution (6-bit to 4-bit mapping).
    - A permutation is applied to the S-box output.
    - The result is XORed with the left half, and the left and right halves are swapped.
- After 16 rounds, the halves are concatenated.

4. Inverse Permutation:

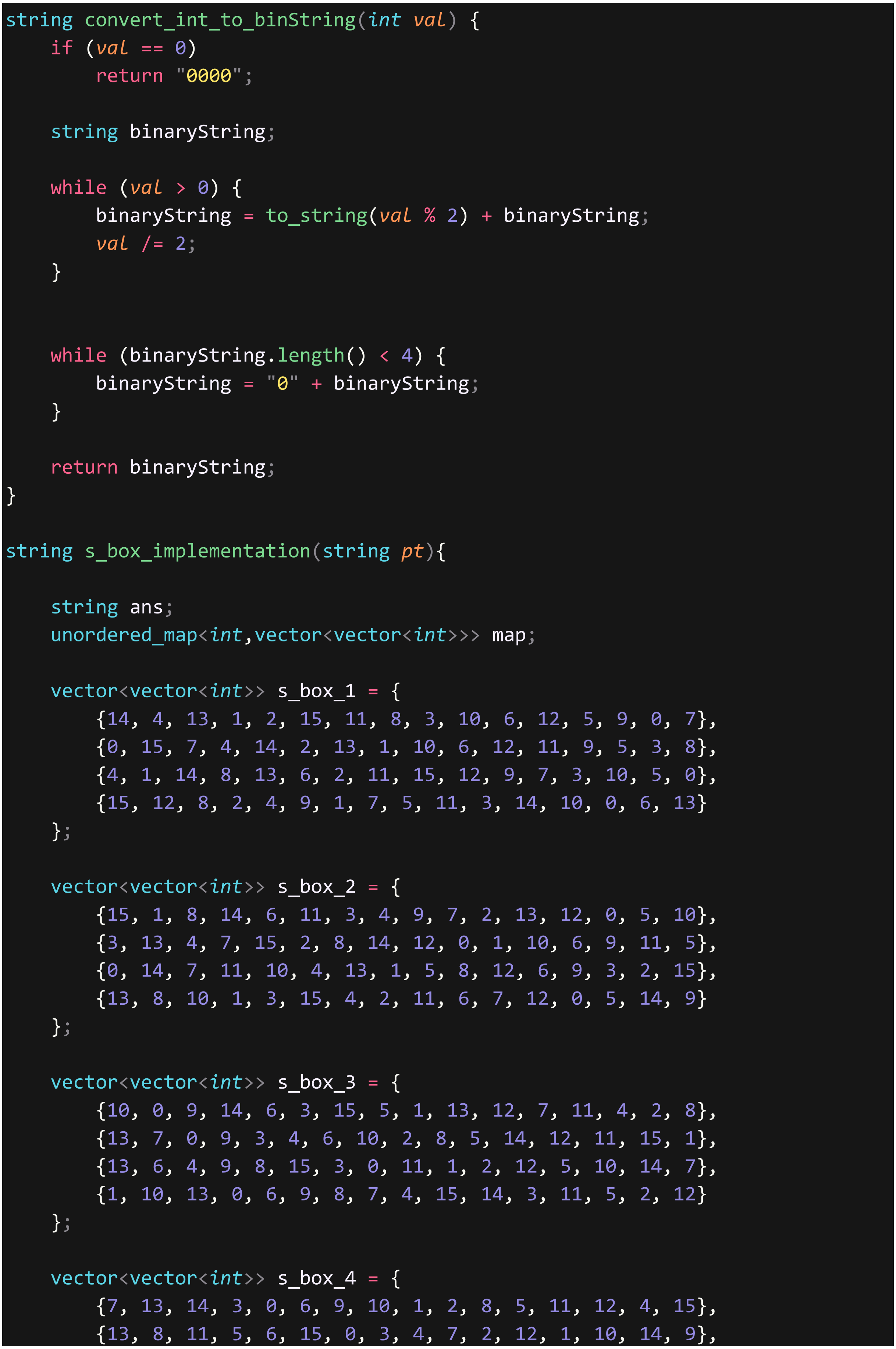
- The output of the Fiestel cipher undergoes an inverse permutation to generate the final 64-bit ciphertext.

5. Decryption:

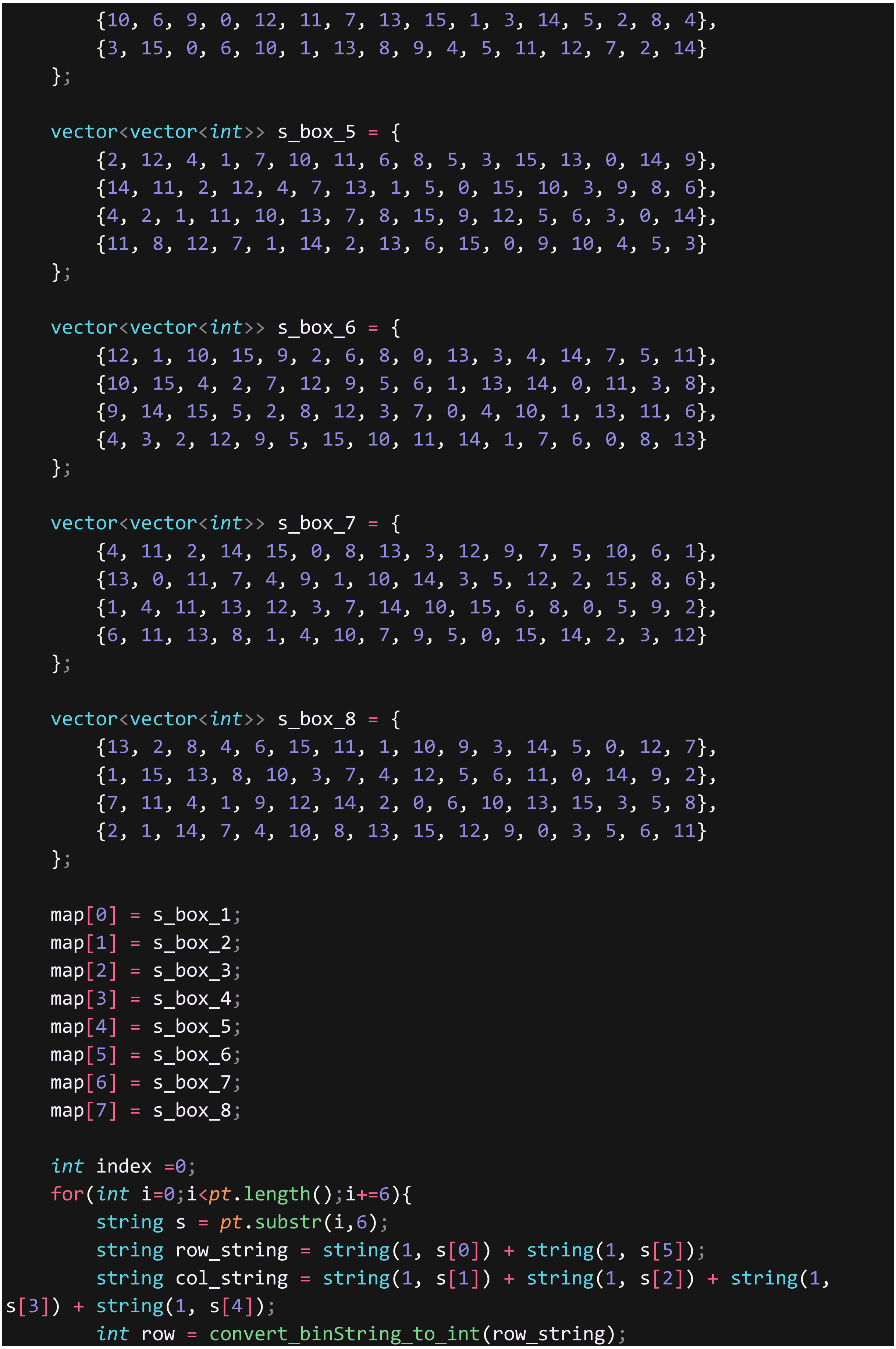
- The ciphertext undergoes the same process in reverse.
- The subkeys are applied in reverse order, and the final output is the decrypted plaintext.

Code –

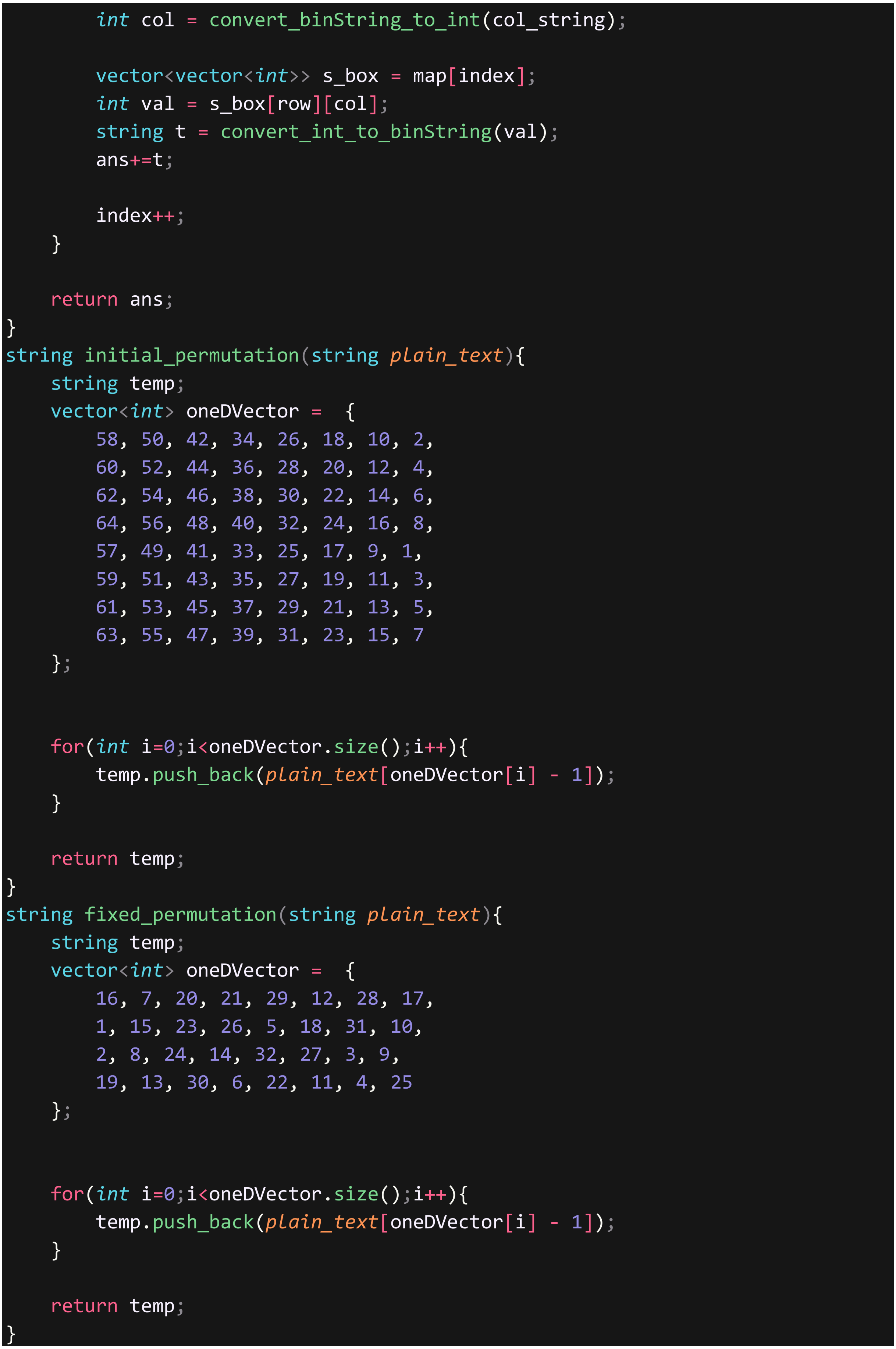
```cpp
// DES
#include<bits/stdc++.h>
using namespace std;
string expansion_permutation(string right){
    vector<int> oneDVector = {
        32, 1, 2, 3, 4, 5,
        4, 5, 6, 7, 8, 9,
        8, 9, 10, 11, 12, 13,
        12, 13, 14, 15, 16, 17,
        16, 17, 18, 19, 20, 21,
        20, 21, 22, 23, 24, 25,
        24, 25, 26, 27, 28, 29,
        28, 29, 30, 31, 32, 1
    };

    string ans;
    for(int i=0;i<oneDVector.size();i++){
        ans.push_back(right[oneDVector[i] - 1]);
    }

    return ans;
}
int convert_binString_to_int(string s) {
    int result = 0;

    for (int i = s.length() - 1; i >= 0; i--) {
        char digit = s[i];
        if (digit == '1') {
            result += pow(2, (s.length() - 1 - i));
        }
    }

    return result;
}
```



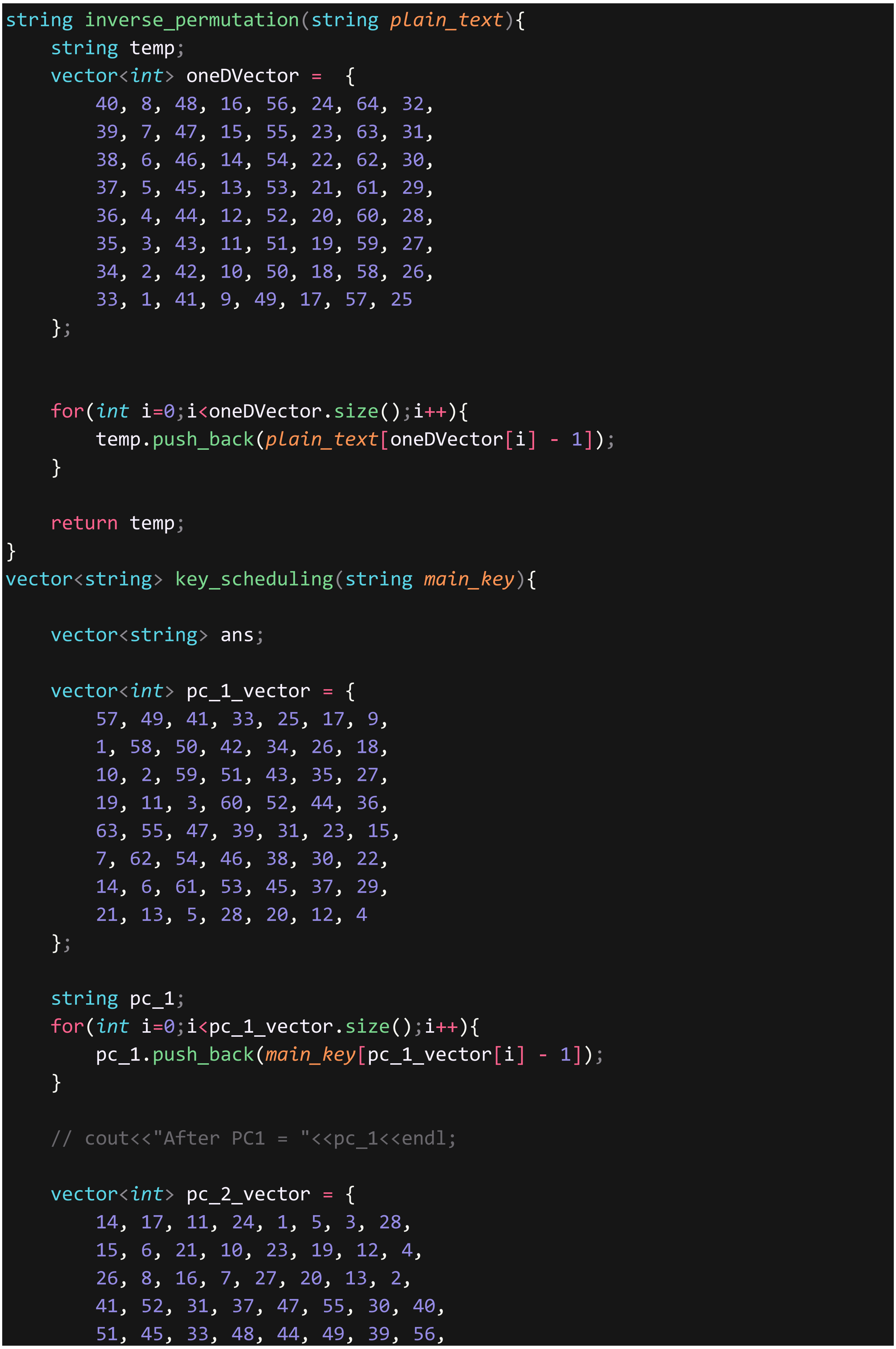
```
string convert_int_to_binString(int val) {
    if (val == 0)
        return "0000";

    string binaryString;

    while (val > 0) {
        binaryString = to_string(val % 2) + binaryString;
        val /= 2;
    }


    while (binaryString.length() < 4) {
        binaryString = "0" + binaryString;
    }

    return binaryString;
}

string s_box_implementation(string pt){

    string ans;
    unordered_map<int,vector<vector<int>>> map;

    vector<vector<int>> s_box_1 = {
        {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}
    };

    vector<vector<int>> s_box_2 = {
        {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}
    };

    vector<vector<int>> s_box_3 = {
        {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}
    };

    vector<vector<int>> s_box_4 = {
        {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},
```



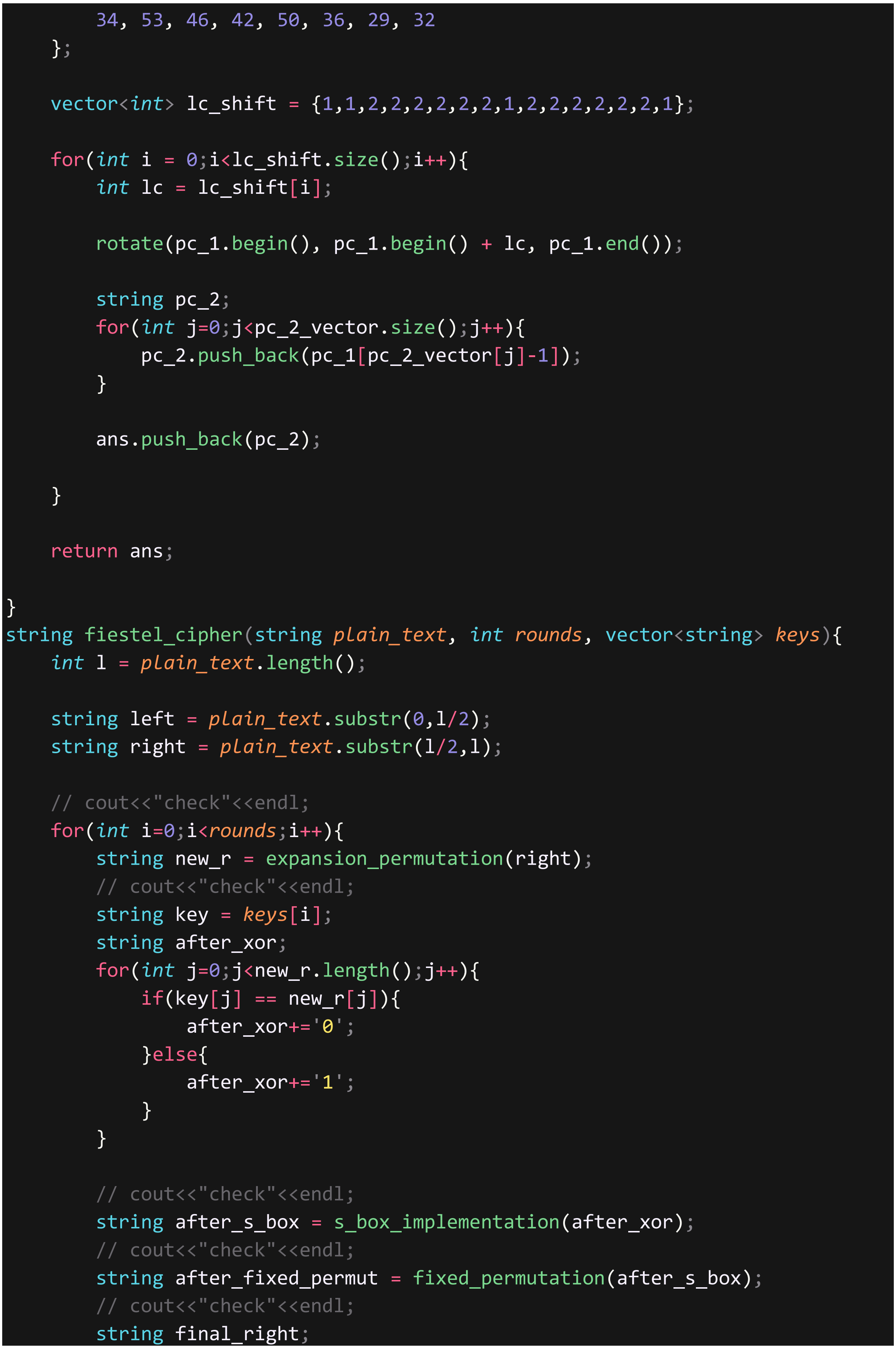
```
        {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},
        {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14}
    };

    vector<vector<int>> s_box_5 = {
        {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}
    };

    vector<vector<int>> s_box_6 = {
        {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}
    };

    vector<vector<int>> s_box_7 = {
        {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}
    };

    vector<vector<int>> s_box_8 = {
        {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}
    };

    map[0] = s_box_1;
    map[1] = s_box_2;
    map[2] = s_box_3;
    map[3] = s_box_4;
    map[4] = s_box_5;
    map[5] = s_box_6;
    map[6] = s_box_7;
    map[7] = s_box_8;

    int index =0;
    for(int i=0;i<pt.length();i+=6){
        string s = pt.substr(i,6);
        string row_string = string(1, s[0]) + string(1, s[5]);
        string col_string = string(1, s[1]) + string(1, s[2]) + string(1,
s[3]) + string(1, s[4]);
        int row = convert_binString_to_int(row_string);
```



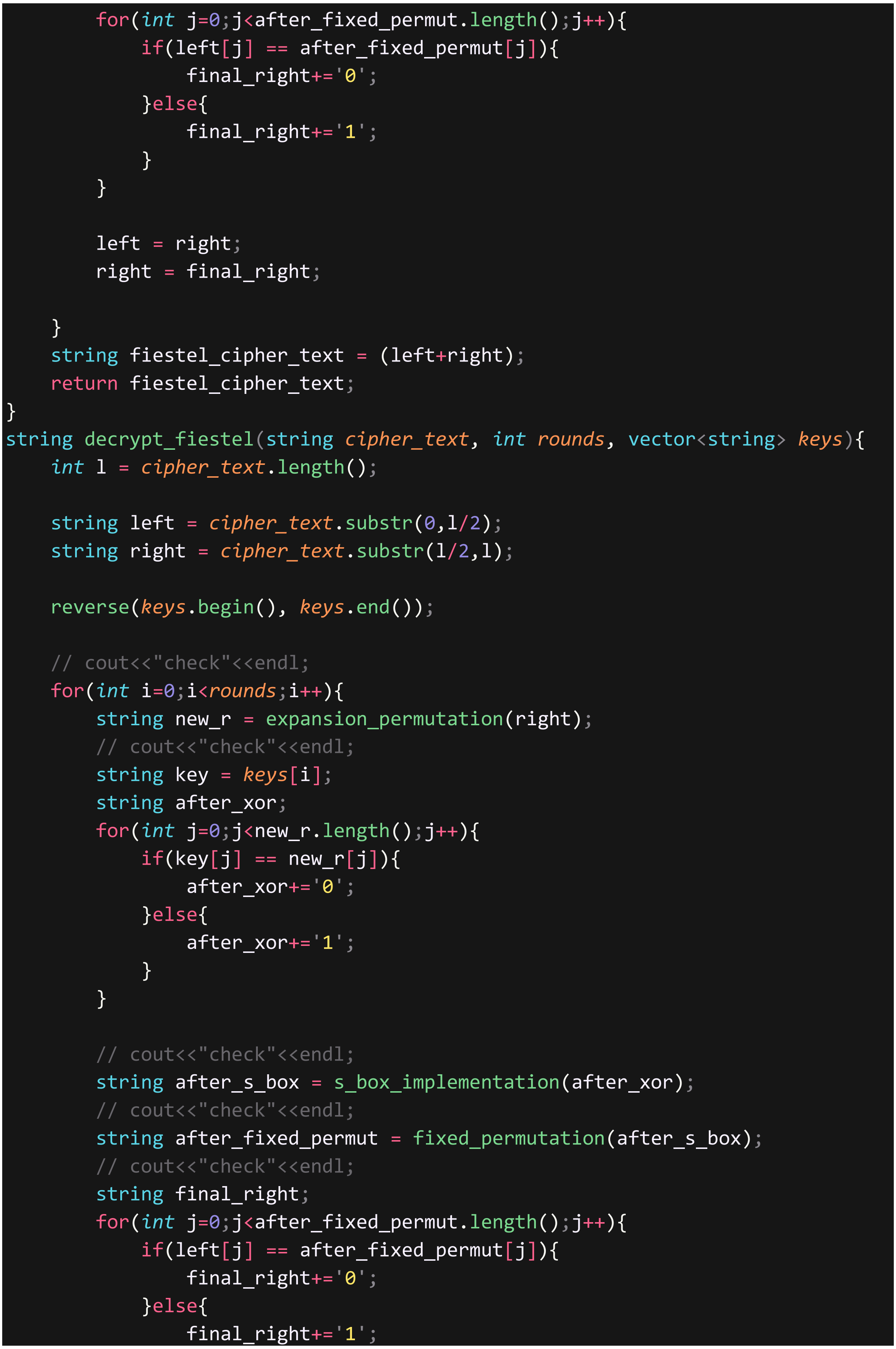
```
        int col = convert_binString_to_int(col_string);

        vector<vector<int>> s_box = map[index];
        int val = s_box[row][col];
        string t = convert_int_to_binString(val);
        ans+=t;

        index++;
    }

    return ans;
}
string initial_permutation(string plain_text){
    string temp;
    vector<int> oneDVector =  {
        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
    };


    for(int i=0;i<oneDVector.size();i++){
        temp.push_back(plain_text[oneDVector[i] - 1]);
    }

    return temp;
}
string fixed_permutation(string plain_text){
    string temp;
    vector<int> oneDVector =  {
        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
    };


    for(int i=0;i<oneDVector.size();i++){
        temp.push_back(plain_text[oneDVector[i] - 1]);
    }

    return temp;
}
```



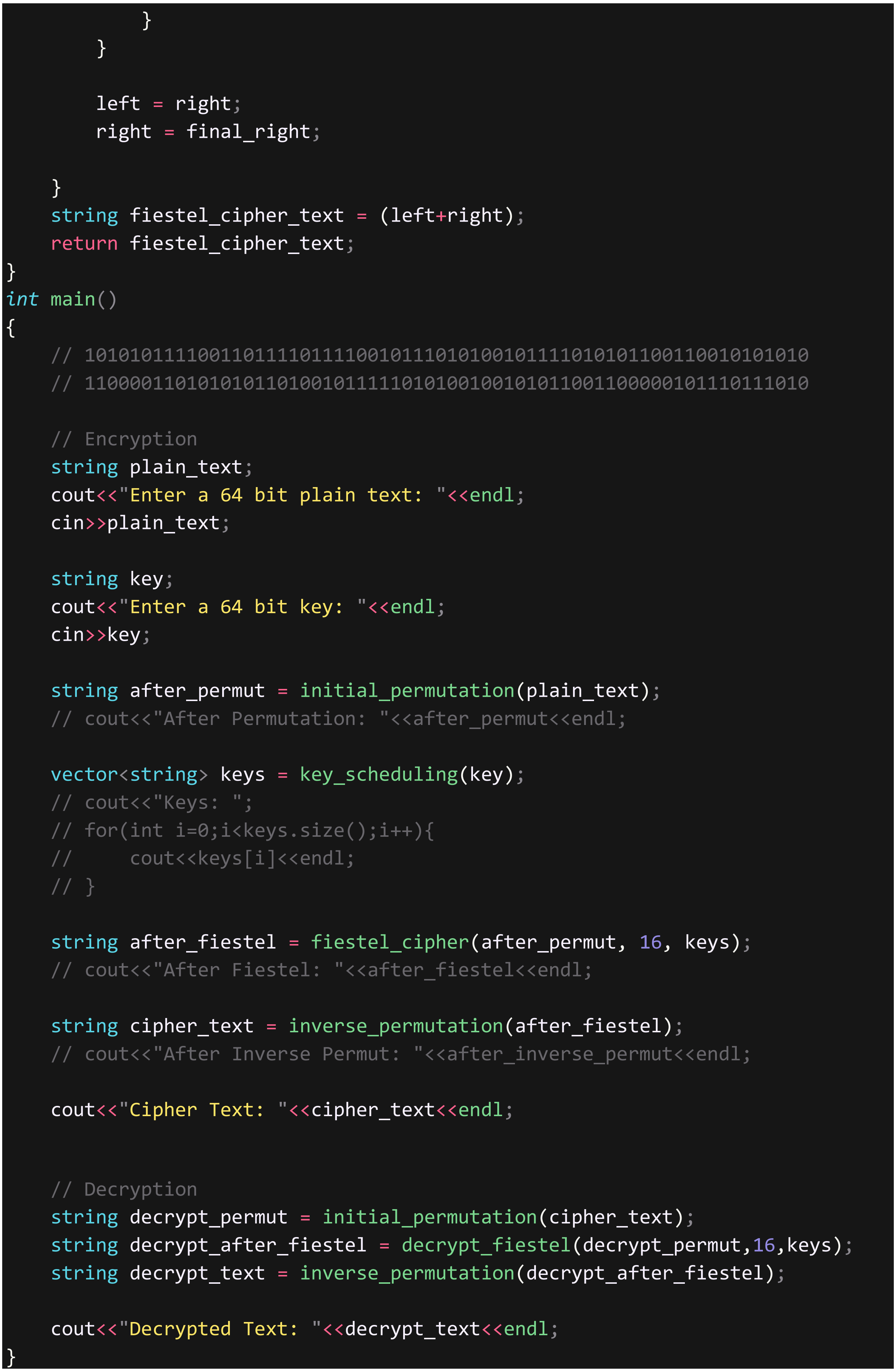
```
string inverse_permutation(string plain_text){
    string temp;
    vector<int> oneDVector =  {
        40, 8, 48, 16, 56, 24, 64, 32,
        39, 7, 47, 15, 55, 23, 63, 31,
        38, 6, 46, 14, 54, 22, 62, 30,
        37, 5, 45, 13, 53, 21, 61, 29,
        36, 4, 44, 12, 52, 20, 60, 28,
        35, 3, 43, 11, 51, 19, 59, 27,
        34, 2, 42, 10, 50, 18, 58, 26,
        33, 1, 41, 9, 49, 17, 57, 25
    };


    for(int i=0;i<oneDVector.size();i++){
        temp.push_back(plain_text[oneDVector[i] - 1]);
    }

    return temp;
}
vector<string> key_scheduling(string main_key){

    vector<string> ans;

    vector<int> pc_1_vector = {
        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
    };

    string pc_1;
    for(int i=0;i<pc_1_vector.size();i++){
        pc_1.push_back(main_key[pc_1_vector[i] - 1]);
    }

    // cout<<"After PC1 = "<<pc_1<<endl;

    vector<int> pc_2_vector = {
        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
    };

    vector<int> lc_shift = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

    for(int i = 0;i<lc_shift.size();i++){
        int lc = lc_shift[i];

        rotate(pc_1.begin(), pc_1.begin() + lc, pc_1.end());

        string pc_2;
        for(int j=0;j<pc_2_vector.size();j++){
            pc_2.push_back(pc_1[pc_2_vector[j]-1]);
        }

        ans.push_back(pc_2);

    }

    return ans;

}
string fiestel_cipher(string plain_text, int rounds, vector<string> keys){
    int l = plain_text.length();

    string left = plain_text.substr(0,l/2);
    string right = plain_text.substr(l/2,l);

    // cout<<"check"<<endl;
    for(int i=0;i<rounds;i++){
        string new_r = expansion_permutation(right);
        // cout<<"check"<<endl;
        string key = keys[i];
        string after_xor;
        for(int j=0;j<new_r.length();j++){
            if(key[j] == new_r[j]){
                after_xor+='0';
            }else{
                after_xor+='1';
            }
        }

        // cout<<"check"<<endl;
        string after_s_box = s_box_implementation(after_xor);
        // cout<<"check"<<endl;
        string after_fixed_permut = fixed_permutation(after_s_box);
        // cout<<"check"<<endl;
        string final_right;
```

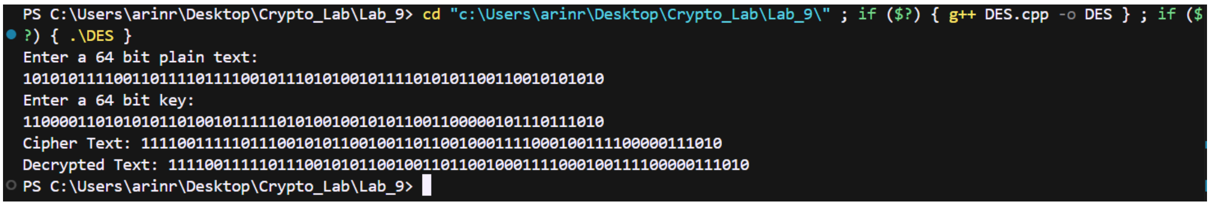


```
        for(int j=0;j<after_fixed_permut.length();j++){
            if(left[j] == after_fixed_permut[j]){
                final_right+='0';
            }else{
                final_right+='1';
            }
        }

        left = right;
        right = final_right;

    }
    string fiestel_cipher_text = (left+right);
    return fiestel_cipher_text;
}
string decrypt_fiestel(string cipher_text, int rounds, vector<string> keys){
    int l = cipher_text.length();

    string left = cipher_text.substr(0,l/2);
    string right = cipher_text.substr(l/2,l);

    reverse(keys.begin(), keys.end());

    // cout<<"check"<<endl;
    for(int i=0;i<rounds;i++){
        string new_r = expansion_permutation(right);
        // cout<<"check"<<endl;
        string key = keys[i];
        string after_xor;
        for(int j=0;j<new_r.length();j++){
            if(key[j] == new_r[j]){
                after_xor+='0';
            }else{
                after_xor+='1';
            }
        }

        // cout<<"check"<<endl;
        string after_s_box = s_box_implementation(after_xor);
        // cout<<"check"<<endl;
        string after_fixed_permut = fixed_permutation(after_s_box);
        // cout<<"check"<<endl;
        string final_right;
        for(int j=0;j<after_fixed_permut.length();j++){
            if(left[j] == after_fixed_permut[j]){
                final_right+='0';
            }else{
                final_right+='1';
```



```
            }
        }

        left = right;
        right = final_right;

    }
    string fiestel_cipher_text = (left+right);
    return fiestel_cipher_text;
}
int main()
{
    // 1010101111001101111011110010111010100101111010101100110010101010
    // 1100001101010101101001011111010100100101011001100000101110111010

    // Encryption
    string plain_text;
    cout<<"Enter a 64 bit plain text: "<<endl;
    cin>>plain_text;

    string key;
    cout<<"Enter a 64 bit key: "<<endl;
    cin>>key;

    string after_permut = initial_permutation(plain_text);
    // cout<<"After Permutation: "<<after_permut<<endl;

    vector<string> keys = key_scheduling(key);
    // cout<<"Keys: ";
    // for(int i=0;i<keys.size();i++){
    //     cout<<keys[i]<<endl;
    // }

    string after_fiestel = fiestel_cipher(after_permut, 16, keys);
    // cout<<"After Fiestel: "<<after_fiestel<<endl;

    string cipher_text = inverse_permutation(after_fiestel);
    // cout<<"After Inverse Permut: "<<after_inverse_permut<<endl;

    cout<<"Cipher Text: "<<cipher_text<<endl;


    // Decryption
    string decrypt_permut = initial_permutation(cipher_text);
    string decrypt_after_fiestel = decrypt_fiestel(decrypt_permut,16,keys);
    string decrypt_text = inverse_permutation(decrypt_after_fiestel);

    cout<<"Decrypted Text: "<<decrypt_text<<endl;
}
```

Output –



```
PS C:\Users\arinr\Desktop\Crypto_Lab\Lab_9> cd "c:\Users\arinr\Desktop\Crypto_Lab\Lab_9\" ; if ($?) { g++ DES.cpp -o DES } ; if ($
?) { .\DES }
Enter a 64 bit plain text:
1010101111001101111011110010111010100101111010101100110010101010
Enter a 64 bit key:
1100001101010101101001011111010100100101011001100000101110111010
Cipher Text: 1111001111101110010101100100110110010001111000100111100000111010
Decrypted Text: 1111001111101110010101100100110110010001111000100111100000111010
PS C:\Users\arinr\Desktop\Crypto_Lab\Lab_9>
```

Q2. Implement the Diffie-Hellman Key Exchange mechanism.

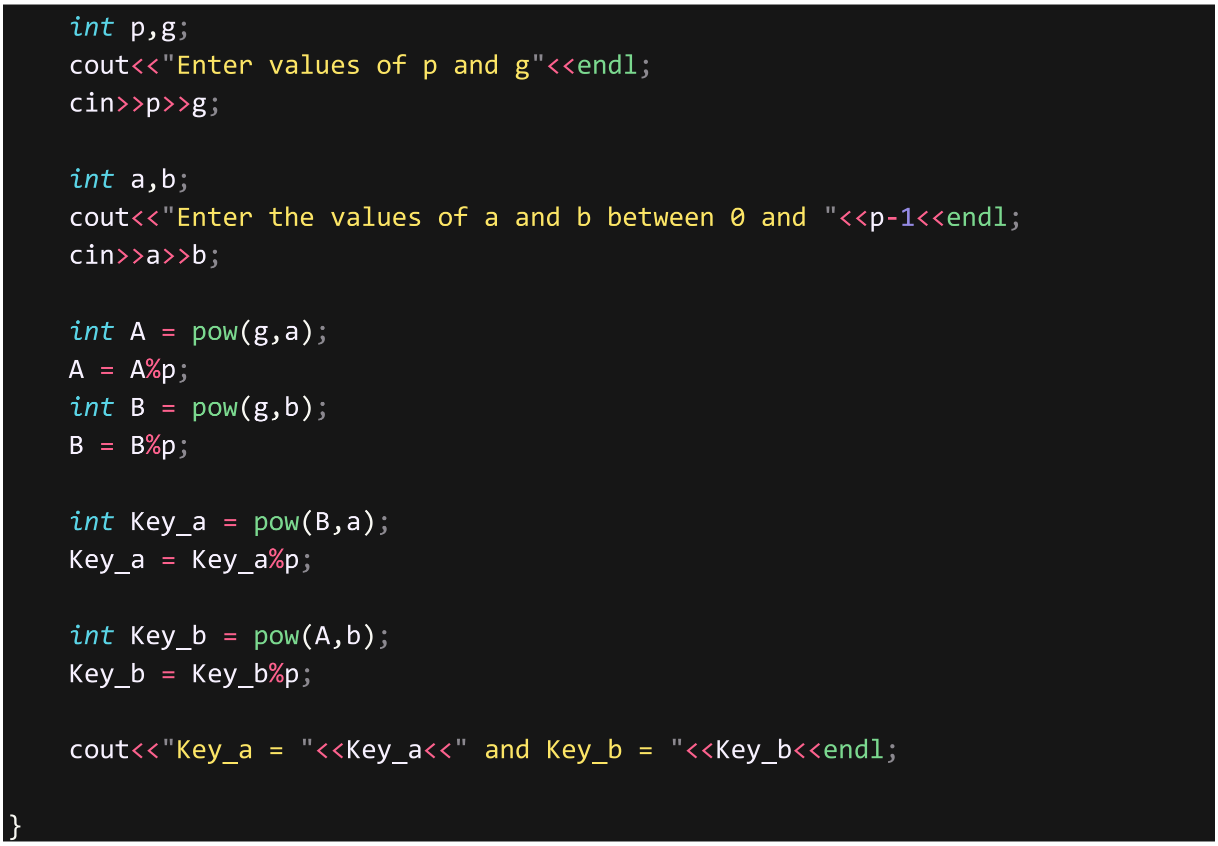
A:

Pseudocode and Explanation –

1. **Input**:
   - The user provides values for p (a prime number) and g (a primitive root modulo p).
   - The user also provides private values a and b, which are kept secret.
2. **Key exchange**:
   - Compute A = (g^a) % p and B = (g^b) % p, where A is sent to B, and B is sent to A.
3. **Shared key computation**:
   - Both parties compute the shared secret key:
     - Party A computes Key_a = (B^a) % p.
     - Party B computes Key_b = (A^b) % p.
4. **Output**:
   - The program outputs the computed keys Key_a and Key_b. These keys should be identical, forming the shared secret.

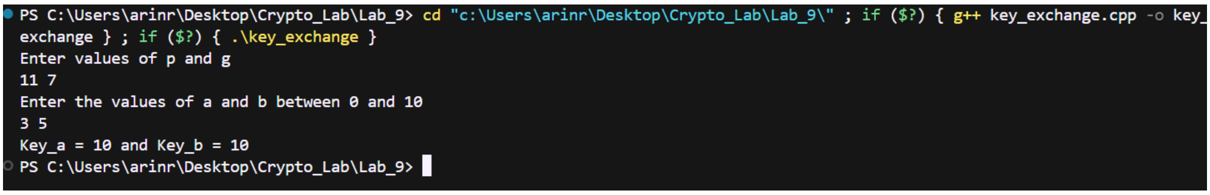
Code –

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
```



```
    int p,g;
    cout<<"Enter values of p and g"<<endl;
    cin>>p>>g;

    int a,b;
    cout<<"Enter the values of a and b between 0 and "<<p-1<<endl;
    cin>>a>>b;

    int A = pow(g,a);
    A = A%p;
    int B = pow(g,b);
    B = B%p;

    int Key_a = pow(B,a);
    Key_a = Key_a%p;

    int Key_b = pow(A,b);
    Key_b = Key_b%p;

    cout<<"Key_a = "<<Key_a<<" and Key_b = "<<Key_b<<endl;

}
```

Output –



```
PS C:\Users\arinr\Desktop\Crypto_Lab\Lab_9> cd "c:\Users\arinr\Desktop\Crypto_Lab\Lab_9\" ; if ($?) { g++ key_exchange.cpp -o key_
exchange } ; if ($?) { .\key_exchange }
Enter values of p and g
11 7
Enter the values of a and b between 0 and 10
3 5
Key_a = 10 and Key_b = 10
PS C:\Users\arinr\Desktop\Crypto_Lab\Lab_9> 
```

Q3. Write a program to implement RSA Algorithm.

A:

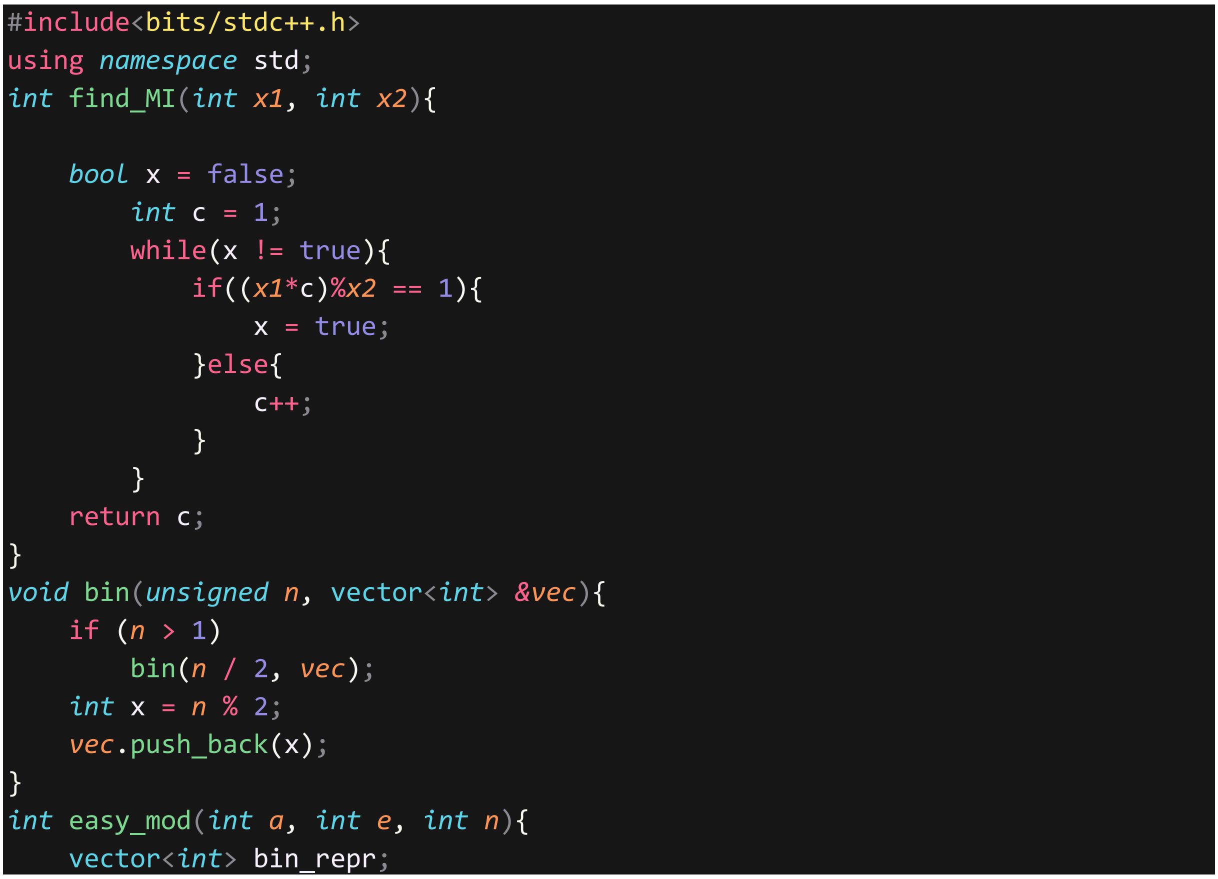
Pseudocode and Explanation –

1. **Prime Numbers (p and q)**:
    - The user inputs two prime numbers p and q. These primes are used to generate the modulus n and the totient phi_n (Euler's Totient Function).
    - n = p * q, and phi_n = (p - 1) * (q - 1).
2. **Public and Private Keys**:

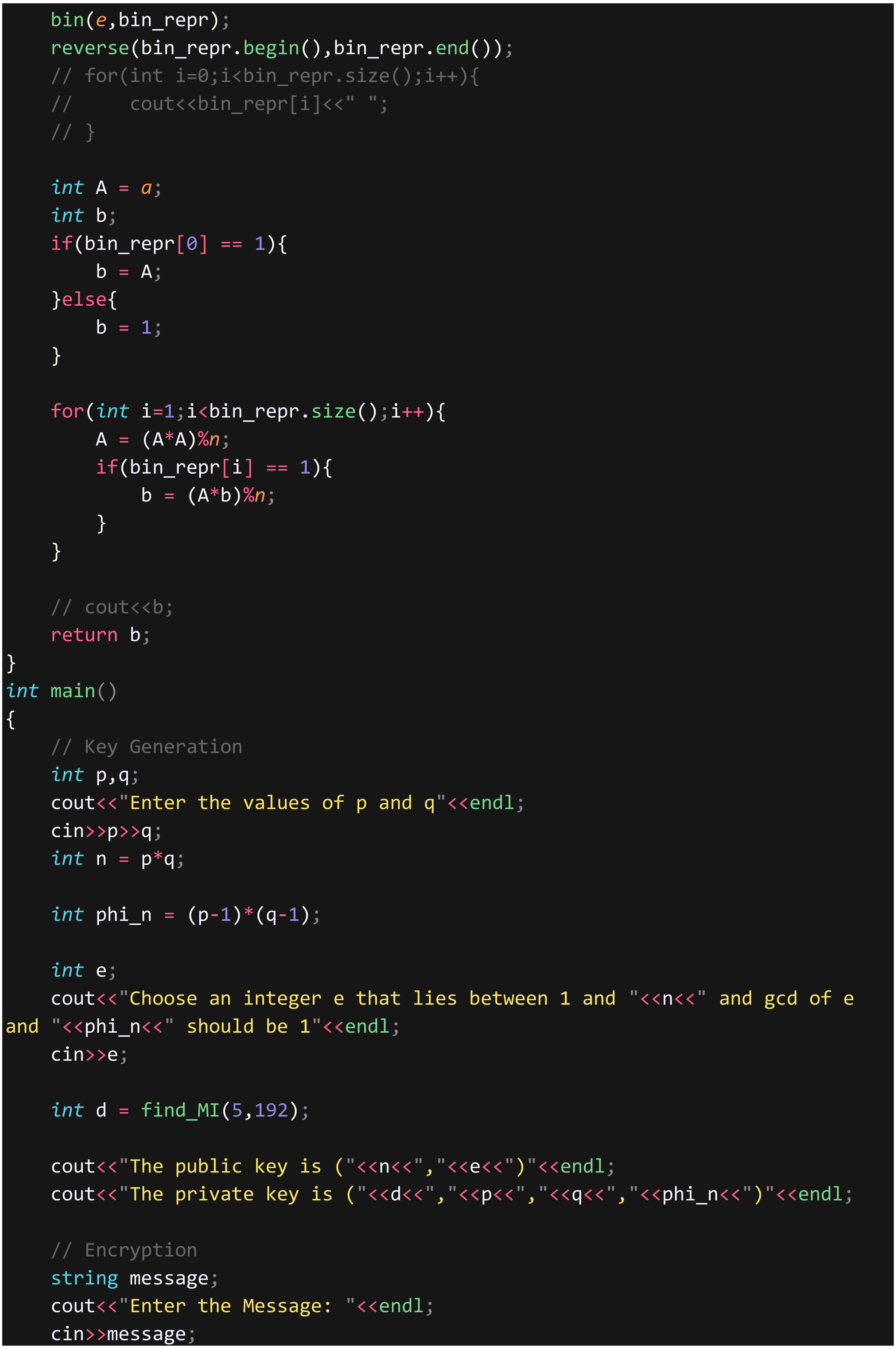
- The public key consists of e and n. The user chooses e such that 1 < e < phi_n and gcd(e, phi_n) = 1.
- The private key is generated by finding d, the modular inverse of e mod phi_n, using the find_MI function.

3. **Encryption and Decryption**:
    - **Encryption**: The message is first converted to an integer format where each character is transformed into a number (A=1, B=2, ..., Z=26), and these numbers are concatenated into a string of integers.
    - The integer message is then encrypted using modular exponentiation: cipher_text = (message^e) % n.
    - **Decryption**: The encrypted message is decrypted using the formula decrypted_text = (cipher_text^d) % n.

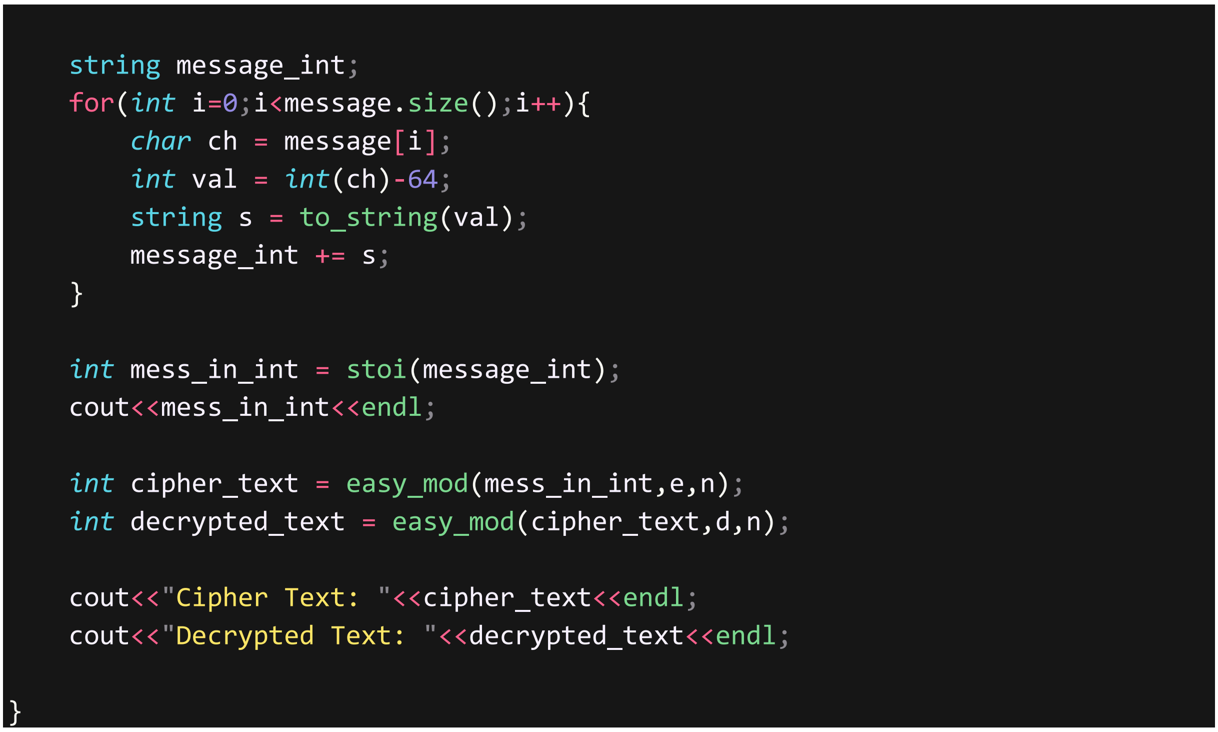
Code –



```
#include<bits/stdc++.h>
using namespace std;
int find_MI(int x1, int x2){

    bool x = false;
        int c = 1;
        while(x != true){
            if((x1*c)%x2 == 1){
                x = true;
            }else{
                c++;
            }
        }
    return c;
}
void bin(unsigned n, vector<int> &vec){
    if (n > 1)
        bin(n / 2, vec);
    int x = n % 2;
    vec.push_back(x);
}
int easy_mod(int a, int e, int n){
    vector<int> bin_repr;
```

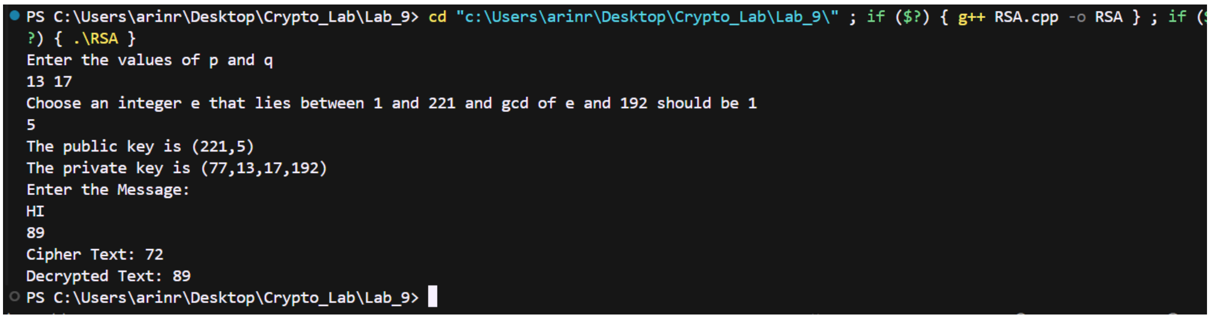


```
    bin(e,bin_repr);
    reverse(bin_repr.begin(),bin_repr.end());
    // for(int i=0;i<bin_repr.size();i++){
    //     cout<<bin_repr[i]<<" ";
    // }

    int A = a;
    int b;
    if(bin_repr[0] == 1){
        b = A;
    }else{
        b = 1;
    }

    for(int i=1;i<bin_repr.size();i++){
        A = (A*A)%n;
        if(bin_repr[i] == 1){
            b = (A*b)%n;
        }
    }

    // cout<<b;
    return b;
}
int main()
{
    // Key Generation
    int p,q;
    cout<<"Enter the values of p and q"<<endl;
    cin>>p>>q;
    int n = p*q;

    int phi_n = (p-1)*(q-1);

    int e;
    cout<<"Choose an integer e that lies between 1 and "<<n<<" and gcd of e
and "<<phi_n<<" should be 1"<<endl;
    cin>>e;

    int d = find_MI(5,192);

    cout<<"The public key is ("<<n<<","<<e<<")"<<endl;
    cout<<"The private key is ("<<d<<","<<p<<","<<q<<","<<phi_n<<")"<<endl;

    // Encryption
    string message;
    cout<<"Enter the Message: "<<endl;
    cin>>message;
```



```
    string message_int;
    for(int i=0;i<message.size();i++){
        char ch = message[i];
        int val = int(ch)-64;
        string s = to_string(val);
        message_int += s;
    }

    int mess_in_int = stoi(message_int);
    cout<<mess_in_int<<endl;

    int cipher_text = easy_mod(mess_in_int,e,n);
    int decrypted_text = easy_mod(cipher_text,d,n);

    cout<<"Cipher Text: "<<cipher_text<<endl;
    cout<<"Decrypted Text: "<<decrypted_text<<endl;

}
```

Output –



```
PS C:\Users\arinr\Desktop\Crypto_Lab\Lab_9> cd "c:\Users\arinr\Desktop\Crypto_Lab\Lab_9\" ; if ($?) { g++ RSA.cpp -o RSA } ; if (
?) { .\RSA }
Enter the values of p and q
13 17
Choose an integer e that lies between 1 and 221 and gcd of e and 192 should be 1
5
The public key is (221,5)
The private key is (77,13,17,192)
Enter the Message:
HI
89
Cipher Text: 72
Decrypted Text: 89
PS C:\Users\arinr\Desktop\Crypto_Lab\Lab_9>
```