# LAB – 11

**Name** : Shubham Pareshbhai Shingala

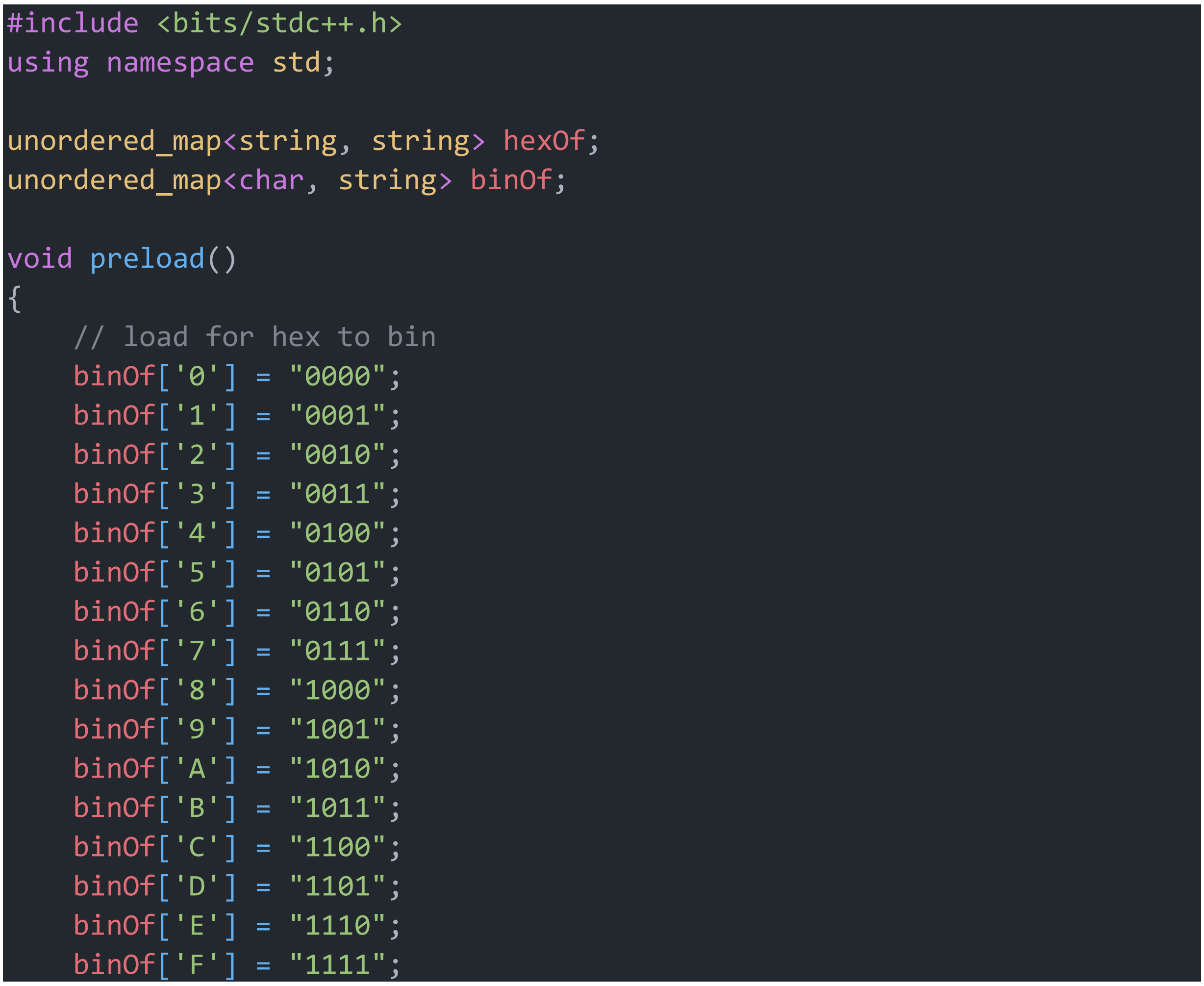
**Roll no.** : CE146

**College ID:** 19CEUOS159

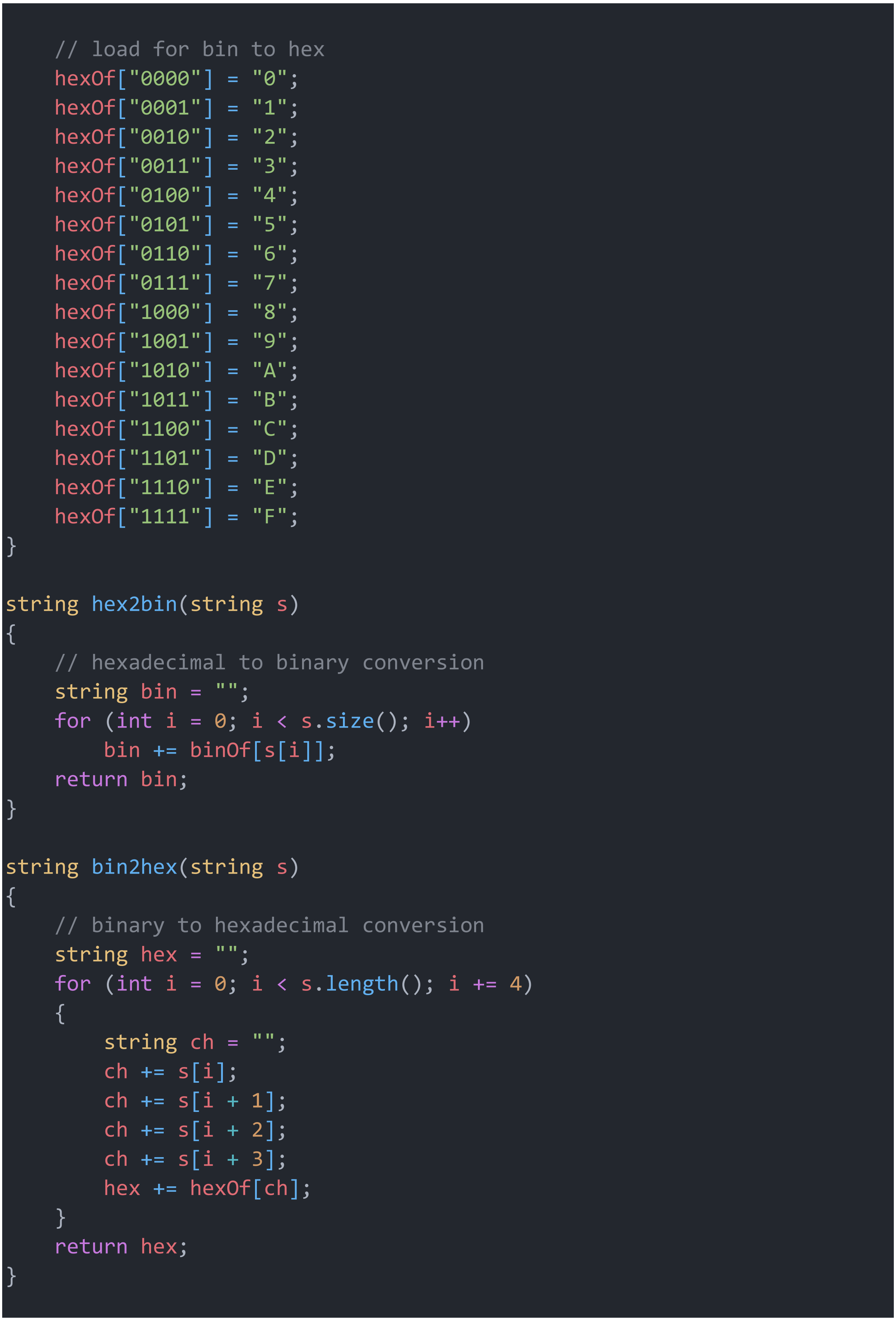
**Aim:** Write a program to implement DES Cipher.

- Encryption
- Decryption
- Key Generation (optional)

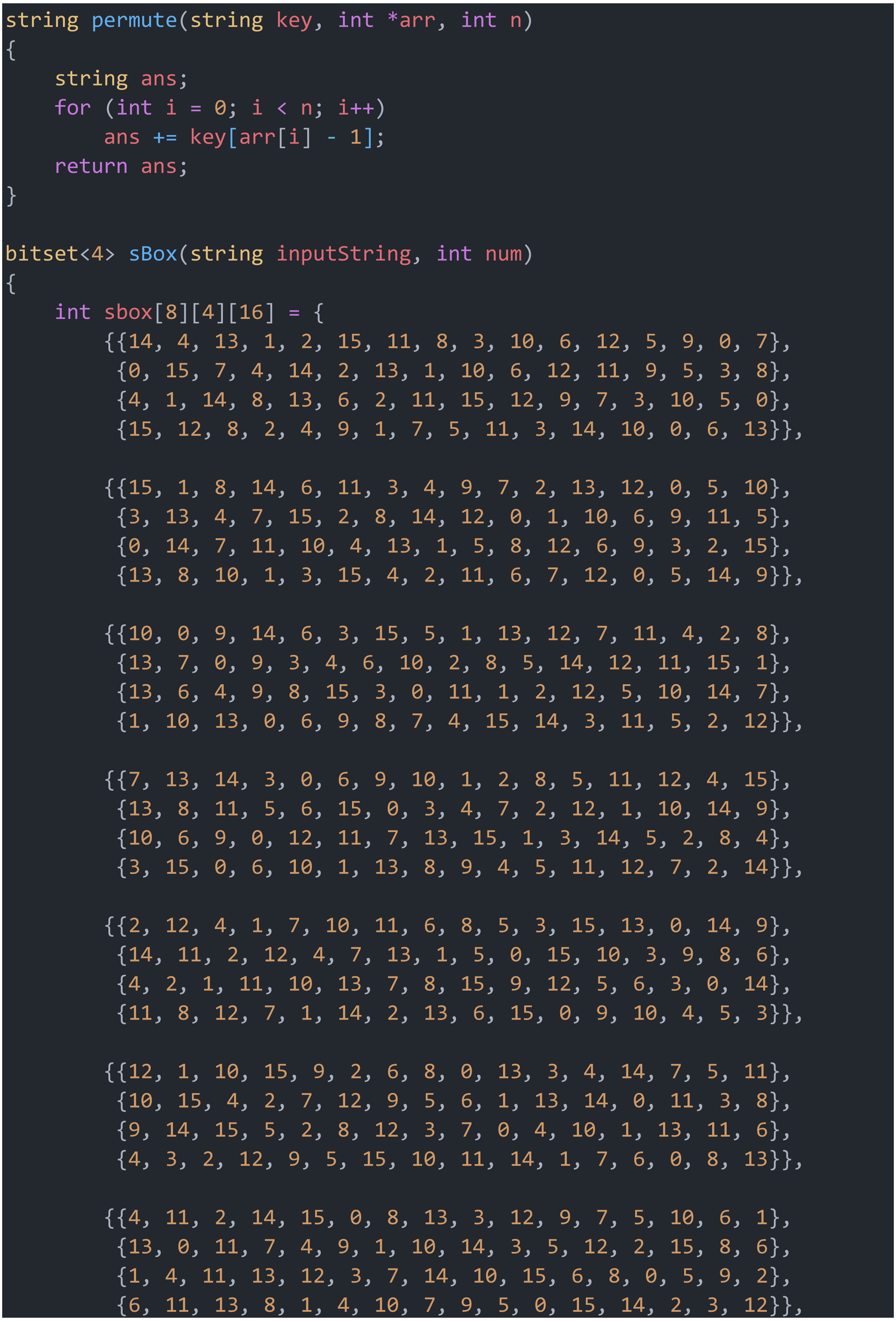
➢ **Source Code:**



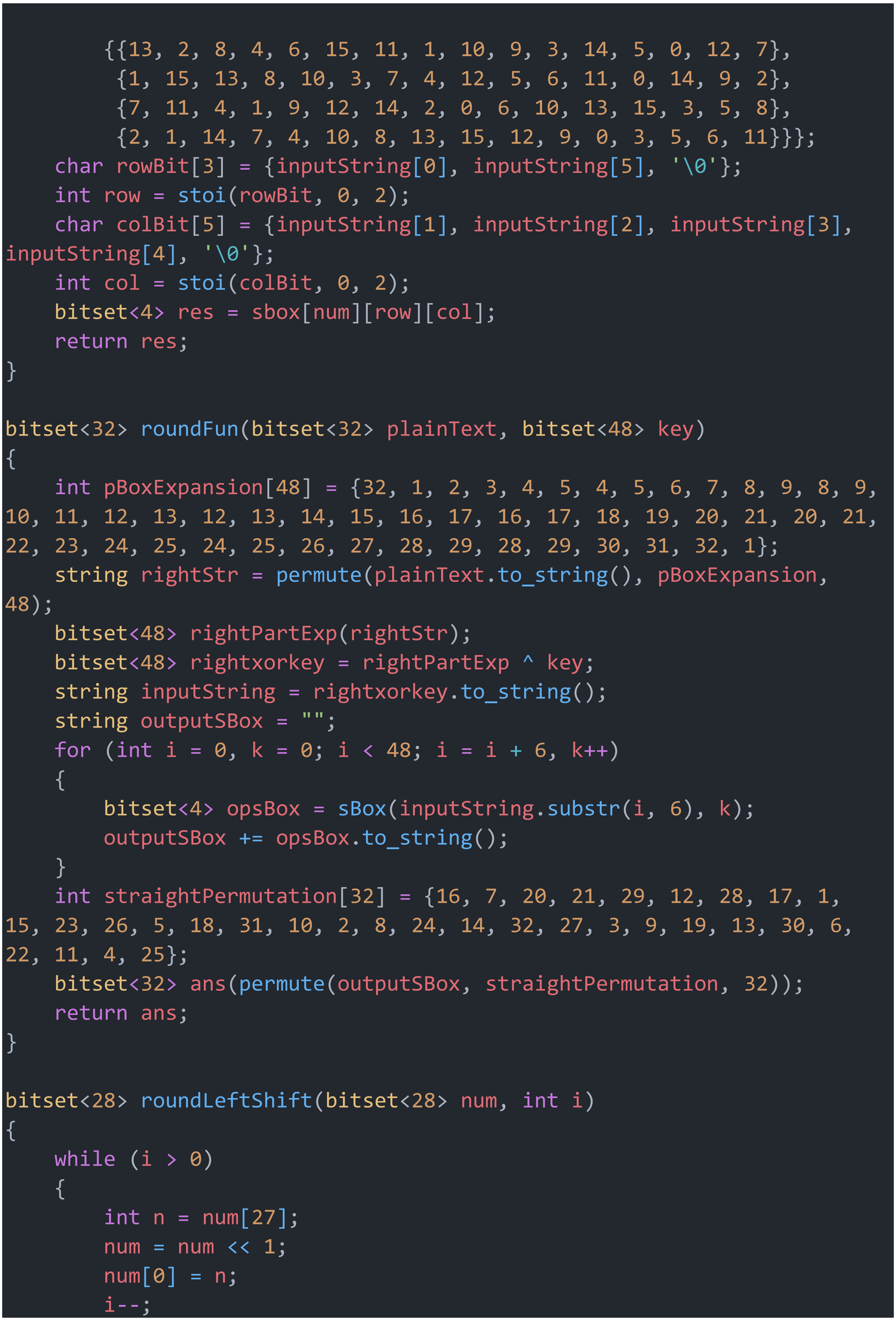
```
#include <bits/stdc++.h>
using namespace std;

unordered_map<string, string> hexOf;
unordered_map<char, string> binOf;

void preload()
{
    // load for hex to bin
    binOf['0'] = "0000";
    binOf['1'] = "0001";
    binOf['2'] = "0010";
    binOf['3'] = "0011";
    binOf['4'] = "0100";
    binOf['5'] = "0101";
    binOf['6'] = "0110";
    binOf['7'] = "0111";
    binOf['8'] = "1000";
    binOf['9'] = "1001";
    binOf['A'] = "1010";
    binOf['B'] = "1011";
    binOf['C'] = "1100";
    binOf['D'] = "1101";
    binOf['E'] = "1110";
    binOf['F'] = "1111";
```



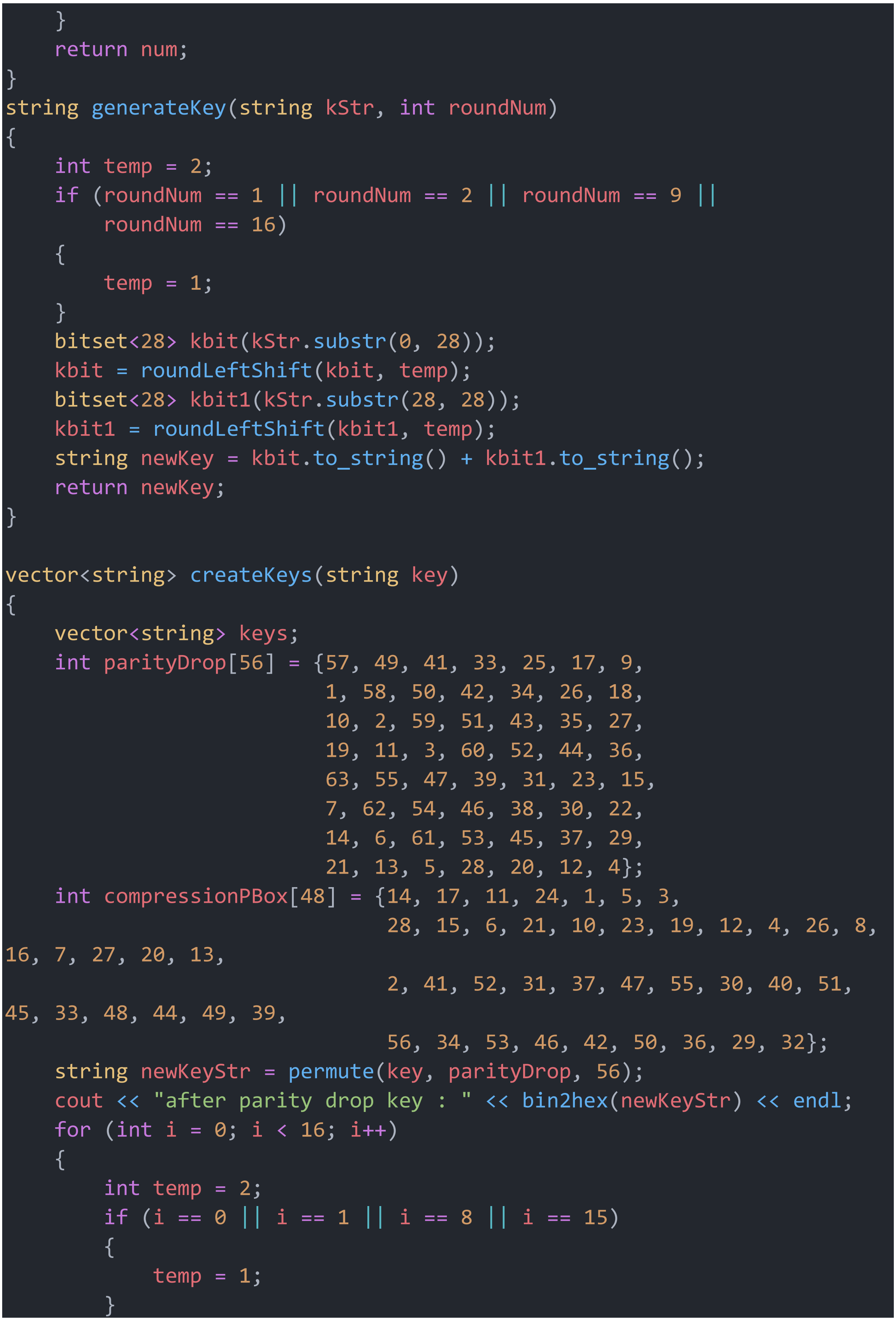
```
    // load for bin to hex
    hexOf["0000"] = "0";
    hexOf["0001"] = "1";
    hexOf["0010"] = "2";
    hexOf["0011"] = "3";
    hexOf["0100"] = "4";
    hexOf["0101"] = "5";
    hexOf["0110"] = "6";
    hexOf["0111"] = "7";
    hexOf["1000"] = "8";
    hexOf["1001"] = "9";
    hexOf["1010"] = "A";
    hexOf["1011"] = "B";
    hexOf["1100"] = "C";
    hexOf["1101"] = "D";
    hexOf["1110"] = "E";
    hexOf["1111"] = "F";
}

string hex2bin(string s)
{
    // hexadecimal to binary conversion
    string bin = "";
    for (int i = 0; i < s.size(); i++)
        bin += binOf[s[i]];
    return bin;
}

string bin2hex(string s)
{
    // binary to hexadecimal conversion
    string hex = "";
    for (int i = 0; i < s.length(); i += 4)
    {
        string ch = "";
        ch += s[i];
        ch += s[i + 1];
        ch += s[i + 2];
        ch += s[i + 3];
        hex += hexOf[ch];
    }
    return hex;
}
```



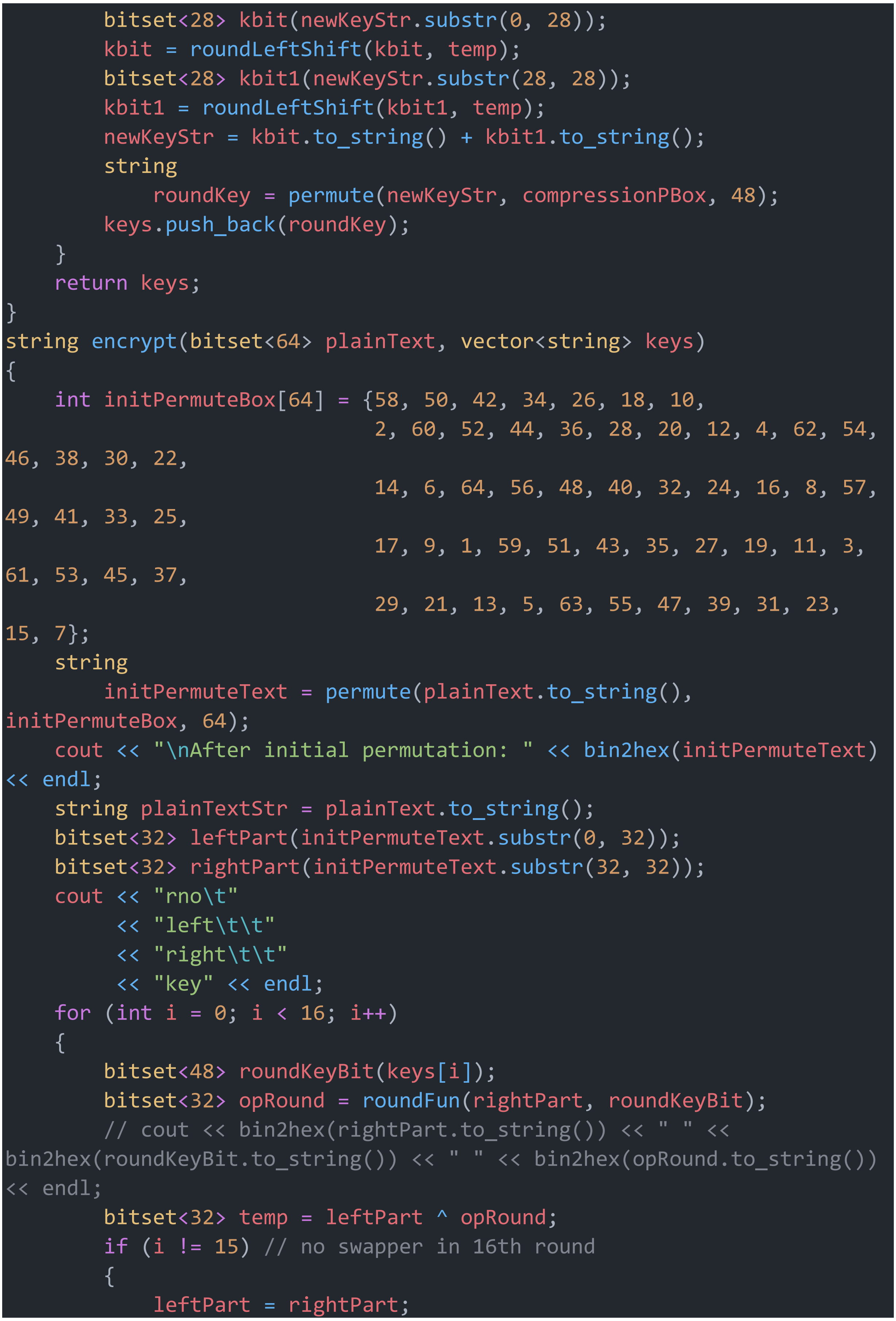
```
string permute(string key, int *arr, int n)
{
    string ans;
    for (int i = 0; i < n; i++)
        ans += key[arr[i] - 1];
    return ans;
}

bitset<4> sBox(string inputString, int num)
{
    int sbox[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}},
```



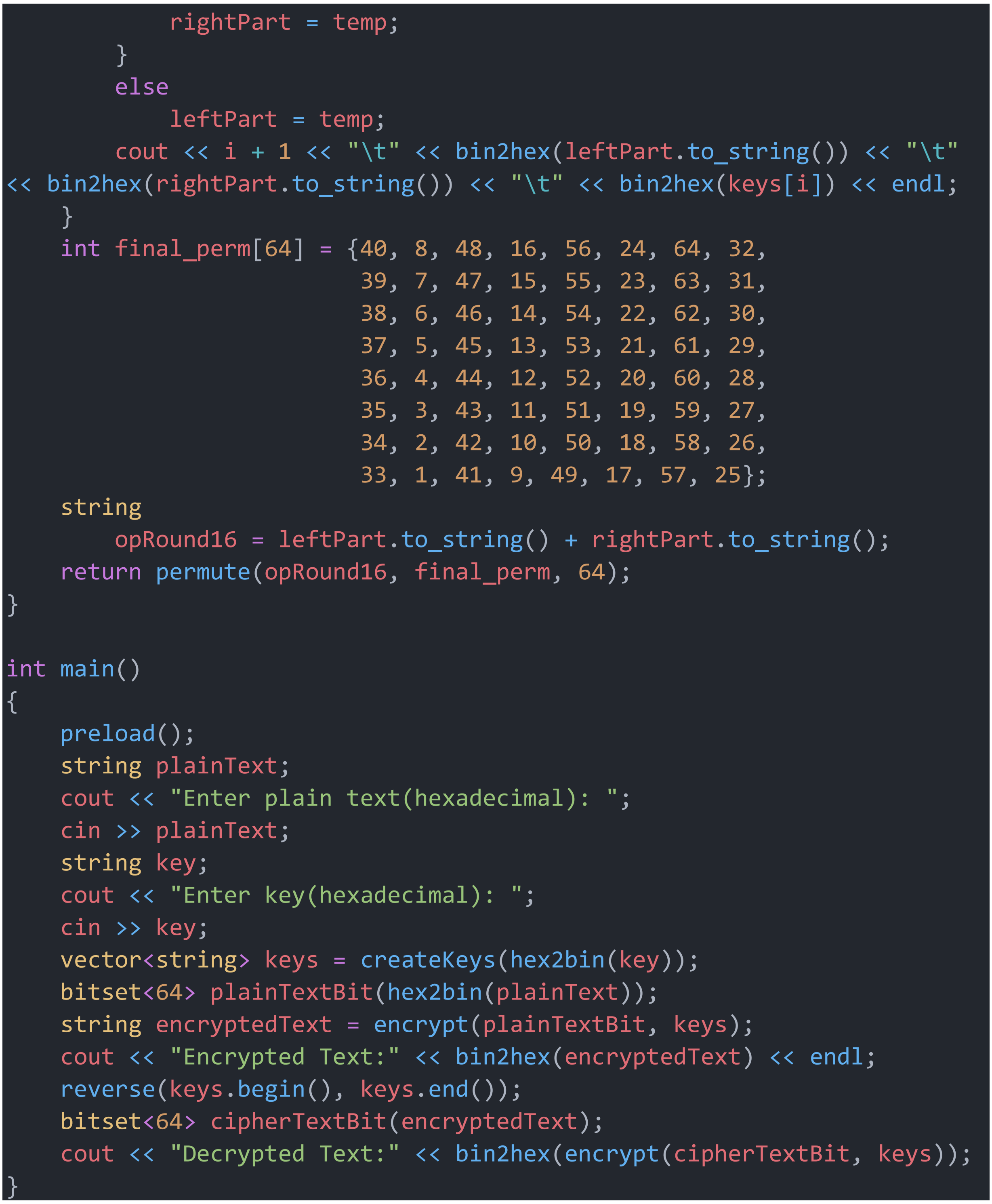
```
        {{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},
         {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2},
         {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},
         {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11}}};
    char rowBit[3] = {inputString[0], inputString[5], '\0'};
    int row = stoi(rowBit, 0, 2);
    char colBit[5] = {inputString[1], inputString[2], inputString[3],
inputString[4], '\0'};
    int col = stoi(colBit, 0, 2);
    bitset<4> res = sbox[num][row][col];
    return res;
}

bitset<32> roundFun(bitset<32> plainText, bitset<48> key)
{
    int pBoxExpansion[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};
    string rightStr = permute(plainText.to_string(), pBoxExpansion,
48);
    bitset<48> rightPartExp(rightStr);
    bitset<48> rightxorkey = rightPartExp ^ key;
    string inputString = rightxorkey.to_string();
    string outputSBox = "";
    for (int i = 0, k = 0; i < 48; i = i + 6, k++)
    {
        bitset<4> opsBox = sBox(inputString.substr(i, 6), k);
        outputSBox += opsBox.to_string();
    }
    int straightPermutation[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};
    bitset<32> ans(permute(outputSBox, straightPermutation, 32));
    return ans;
}

bitset<28> roundLeftShift(bitset<28> num, int i)
{
    while (i > 0)
    {
        int n = num[27];
        num = num << 1;
        num[0] = n;
        i--;
```



```
    }
    return num;
}
string generateKey(string kStr, int roundNum)
{
    int temp = 2;
    if (roundNum == 1 || roundNum == 2 || roundNum == 9 ||
        roundNum == 16)
    {
        temp = 1;
    }
    bitset<28> kbit(kStr.substr(0, 28));
    kbit = roundLeftShift(kbit, temp);
    bitset<28> kbit1(kStr.substr(28, 28));
    kbit1 = roundLeftShift(kbit1, temp);
    string newKey = kbit.to_string() + kbit1.to_string();
    return newKey;
}

vector<string> createKeys(string key)
{
    vector<string> keys;
    int parityDrop[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};
    int compressionPBox[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};
    string newKeyStr = permute(key, parityDrop, 56);
    cout << "after parity drop key : " << bin2hex(newKeyStr) << endl;
    for (int i = 0; i < 16; i++)
    {
        int temp = 2;
        if (i == 0 || i == 1 || i == 8 || i == 15)
        {
            temp = 1;
        }
```

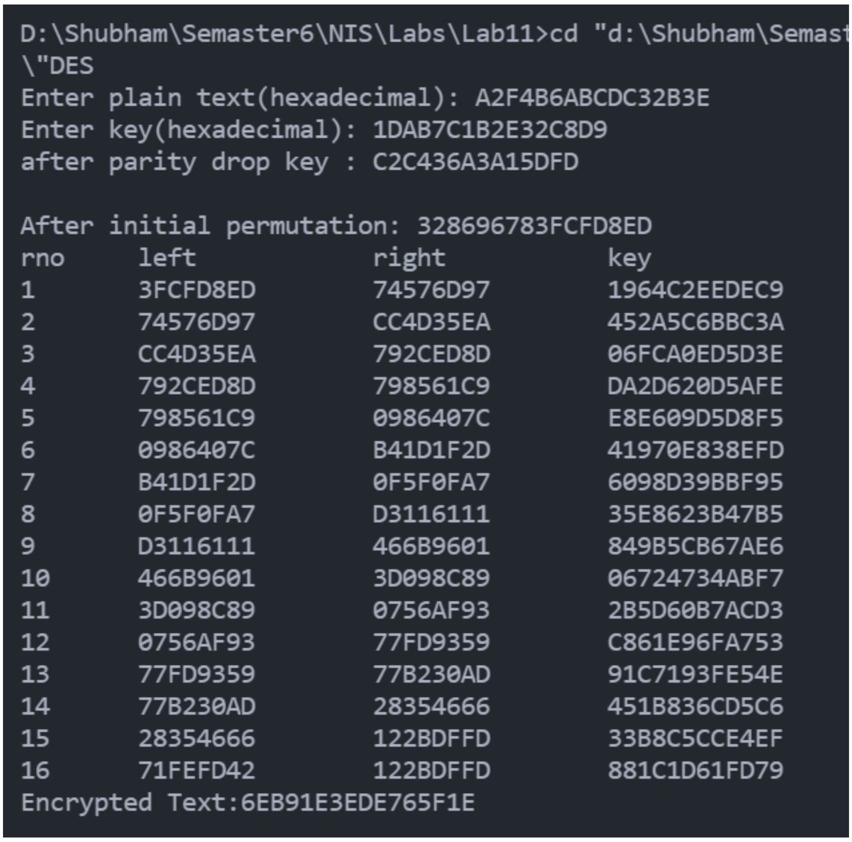


```
        bitset<28> kbit(newKeyStr.substr(0, 28));
        kbit = roundLeftShift(kbit, temp);
        bitset<28> kbit1(newKeyStr.substr(28, 28));
        kbit1 = roundLeftShift(kbit1, temp);
        newKeyStr = kbit.to_string() + kbit1.to_string();
        string
            roundKey = permute(newKeyStr, compressionPBox, 48);
        keys.push_back(roundKey);
    }
    return keys;
}
string encrypt(bitset<64> plainText, vector<string> keys)
{
    int initPermuteBox[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};
    string
        initPermuteText = permute(plainText.to_string(),
initPermuteBox, 64);
    cout << "\nAfter initial permutation: " << bin2hex(initPermuteText)
<< endl;
    string plainTextStr = plainText.to_string();
    bitset<32> leftPart(initPermuteText.substr(0, 32));
    bitset<32> rightPart(initPermuteText.substr(32, 32));
    cout << "rno\t"
         << "left\t\t"
         << "right\t\t"
         << "key" << endl;
    for (int i = 0; i < 16; i++)
    {
        bitset<48> roundKeyBit(keys[i]);
        bitset<32> opRound = roundFun(rightPart, roundKeyBit);
        // cout << bin2hex(rightPart.to_string()) << " " <<
bin2hex(roundKeyBit.to_string()) << " " << bin2hex(opRound.to_string())
<< endl;
        bitset<32> temp = leftPart ^ opRound;
        if (i != 15) // no swapper in 16th round
        {
            leftPart = rightPart;
```

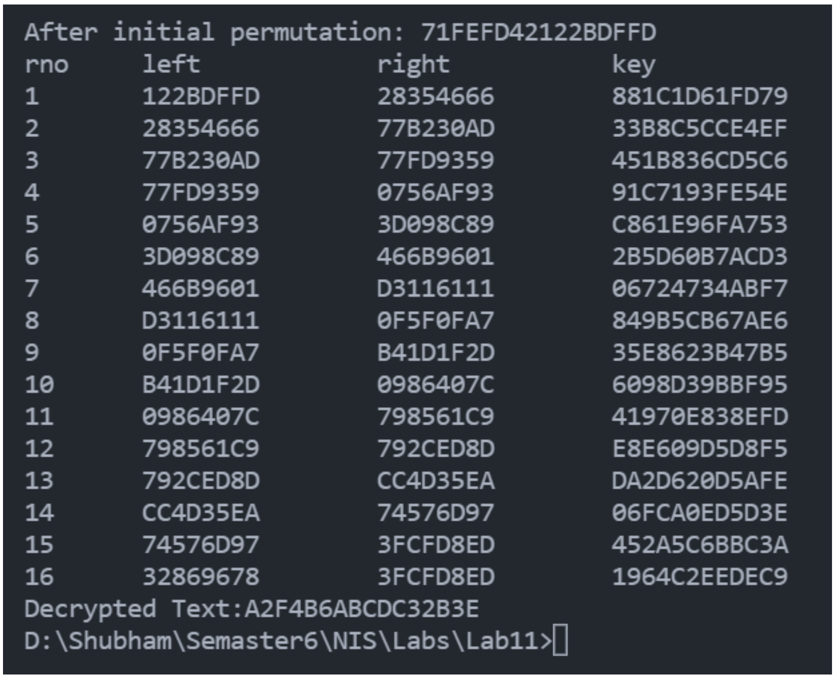


```
            rightPart = temp;
        }
        else
            leftPart = temp;
        cout << i + 1 << "\t" << bin2hex(leftPart.to_string()) << "\t"
<< bin2hex(rightPart.to_string()) << "\t" << bin2hex(keys[i]) << endl;
    }
    int final_perm[64] = {40, 8, 48, 16, 56, 24, 64, 32,
                          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};
    string
        opRound16 = leftPart.to_string() + rightPart.to_string();
    return permute(opRound16, final_perm, 64);
}

int main()
{
    preload();
    string plainText;
    cout << "Enter plain text(hexadecimal): ";
    cin >> plainText;
    string key;
    cout << "Enter key(hexadecimal): ";
    cin >> key;
    vector<string> keys = createKeys(hex2bin(key));
    bitset<64> plainTextBit(hex2bin(plainText));
    string encryptedText = encrypt(plainTextBit, keys);
    cout << "Encrypted Text:" << bin2hex(encryptedText) << endl;
    reverse(keys.begin(), keys.end());
    bitset<64> cipherTextBit(encryptedText);
    cout << "Decrypted Text:" << bin2hex(encrypt(cipherTextBit, keys));
}
```

## ➢ Test Case – 1:



```
D:\Shubham\Semaster6\NIS\Labs\Lab11>cd "d:\Shubham\Semast
\"DES
Enter plain text(hexadecimal): A2F4B6ABCDC32B3E
Enter key(hexadecimal): 1DAB7C1B2E32C8D9
after parity drop key : C2C436A3A15DFD

After initial permutation: 328696783FCFD8ED
rno     left            right           key
1       3FCFD8ED        74576D97        1964C2EEDEC9
2       74576D97        CC4D35EA        452A5C6BBC3A
3       CC4D35EA        792CED8D        06FCA0ED5D3E
4       792CED8D        798561C9        DA2D620D5AFE
5       798561C9        0986407C        E8E609D5D8F5
6       0986407C        B41D1F2D        41970E838EFD
7       B41D1F2D        0F5F0FA7        6098D39BBF95
8       0F5F0FA7        D3116111        35E8623B47B5
9       D3116111        466B9601        849B5CB67AE6
10      466B9601        3D098C89        06724734ABF7
11      3D098C89        0756AF93        2B5D60B7ACD3
12      0756AF93        77FD9359        C861E96FA753
13      77FD9359        77B230AD        91C7193FE54E
14      77B230AD        28354666        451B836CD5C6
15      28354666        122BDFFD        33B8C5CCE4EF
16      71FEFD42        122BDFFD        881C1D61FD79
Encrypted Text:6EB91E3EDE765F1E
```



```
After initial permutation: 71FEFD42122BDFFD
rno     left            right           key
1       122BDFFD        28354666        881C1D61FD79
2       28354666        77B230AD        33B8C5CCE4EF
3       77B230AD        77FD9359        451B836CD5C6
4       77FD9359        0756AF93        91C7193FE54E
5       0756AF93        3D098C89        C861E96FA753
6       3D098C89        466B9601        2B5D60B7ACD3
7       466B9601        D3116111        06724734ABF7
8       D3116111        0F5F0FA7        849B5CB67AE6
9       0F5F0FA7        B41D1F2D        35E8623B47B5
10      B41D1F2D        0986407C        6098D39BBF95
11      0986407C        798561C9        41970E838EFD
12      798561C9        792CED8D        E8E609D5D8F5
13      792CED8D        CC4D35EA        DA2D620D5AFE
14      CC4D35EA        74576D97        06FCA0ED5D3E
15      74576D97        3FCFD8ED        452A5C6BBC3A
16      32869678        3FCFD8ED        1964C2EEDEC9
Decrypted Text:A2F4B6ABCDC32B3E
D:\Shubham\Semaster6\NIS\Labs\Lab11>
```