Lab 11

Name: Vaghani Smit Dhirubhai

Roll No: CE169

College Id: 19CEUEG022

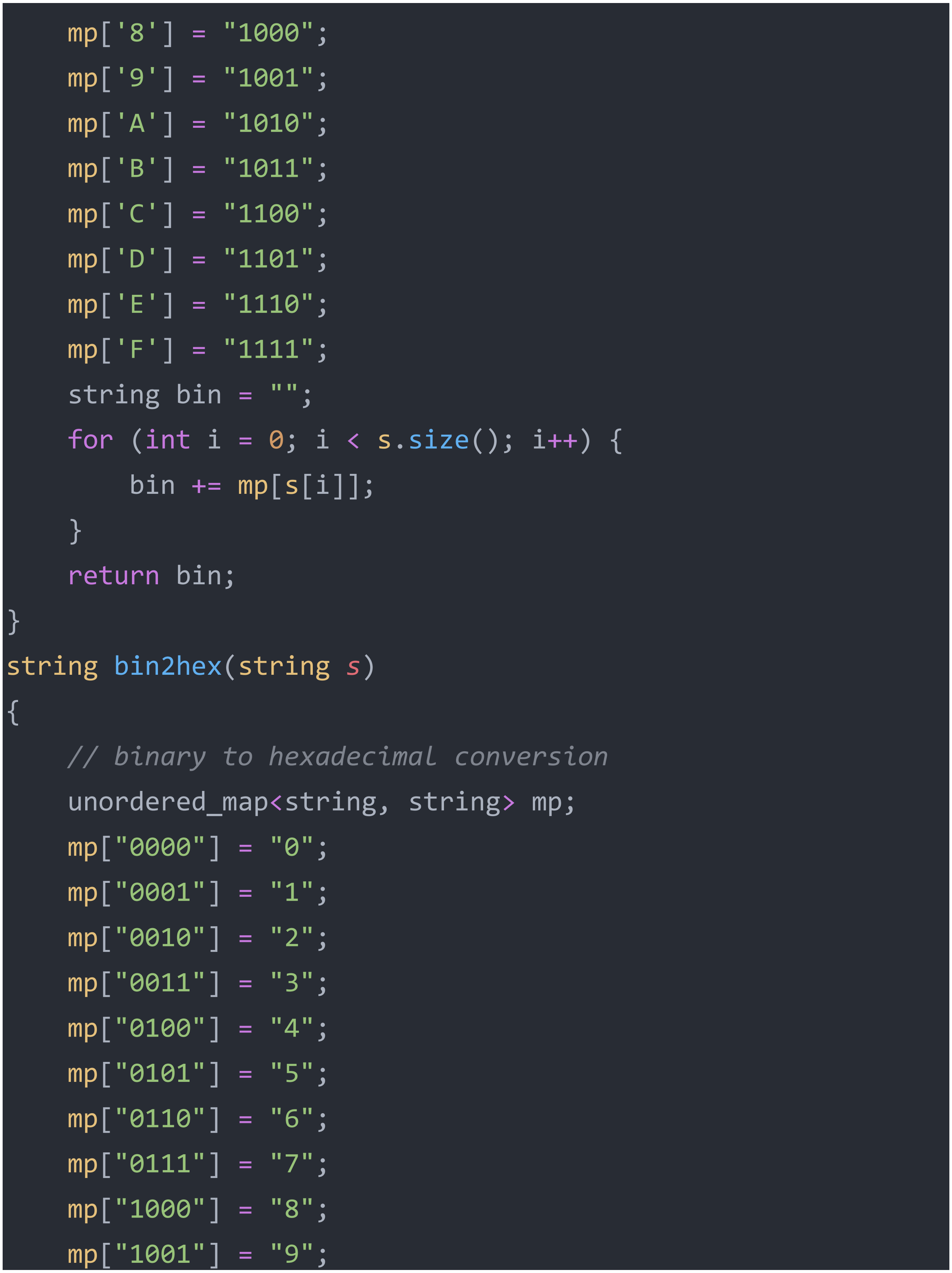
Aim: Write a program to implement DES Cipher.

- Encryption,
- Decryption,
- Key Generation (optional)

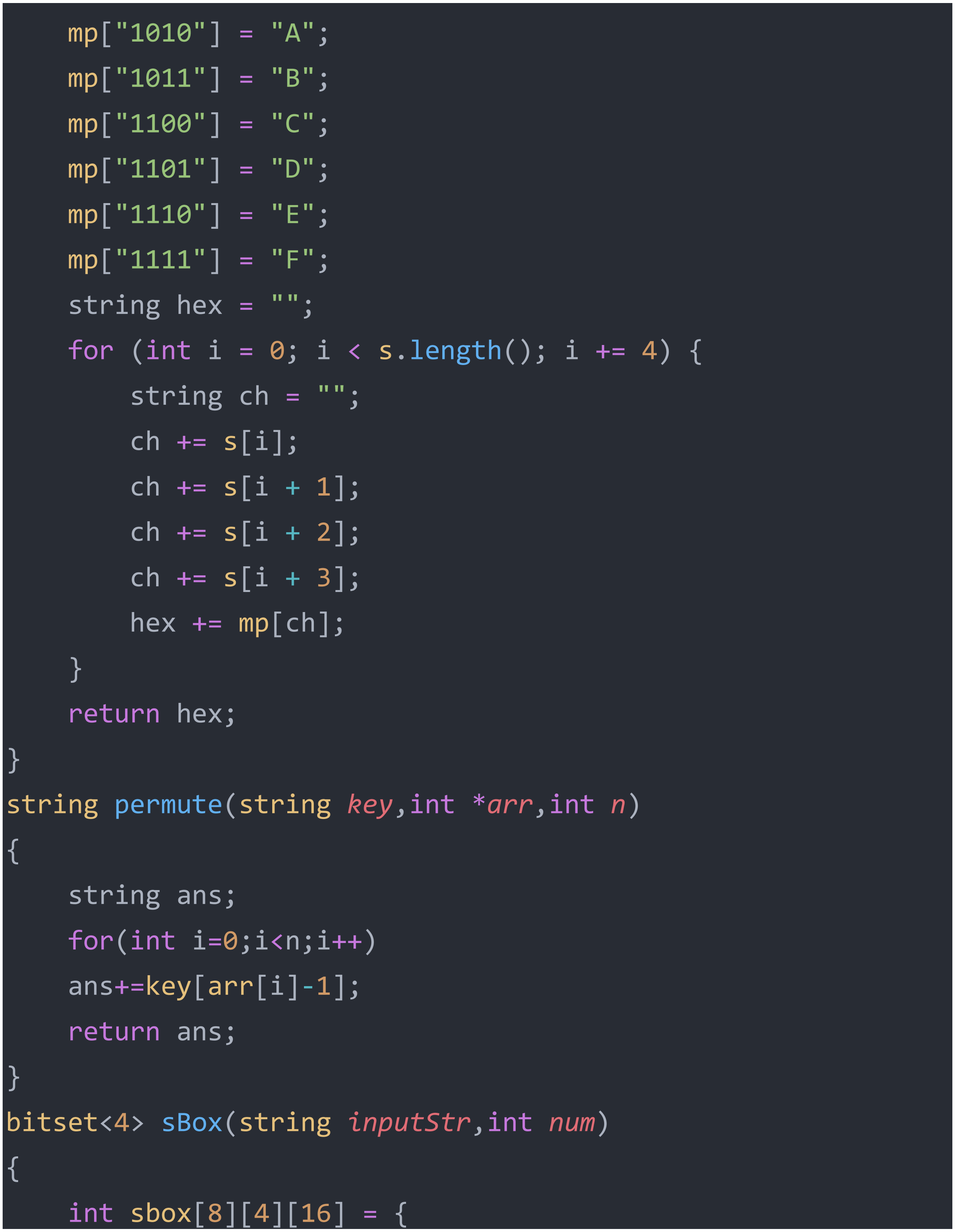
DES Cipher

Code:

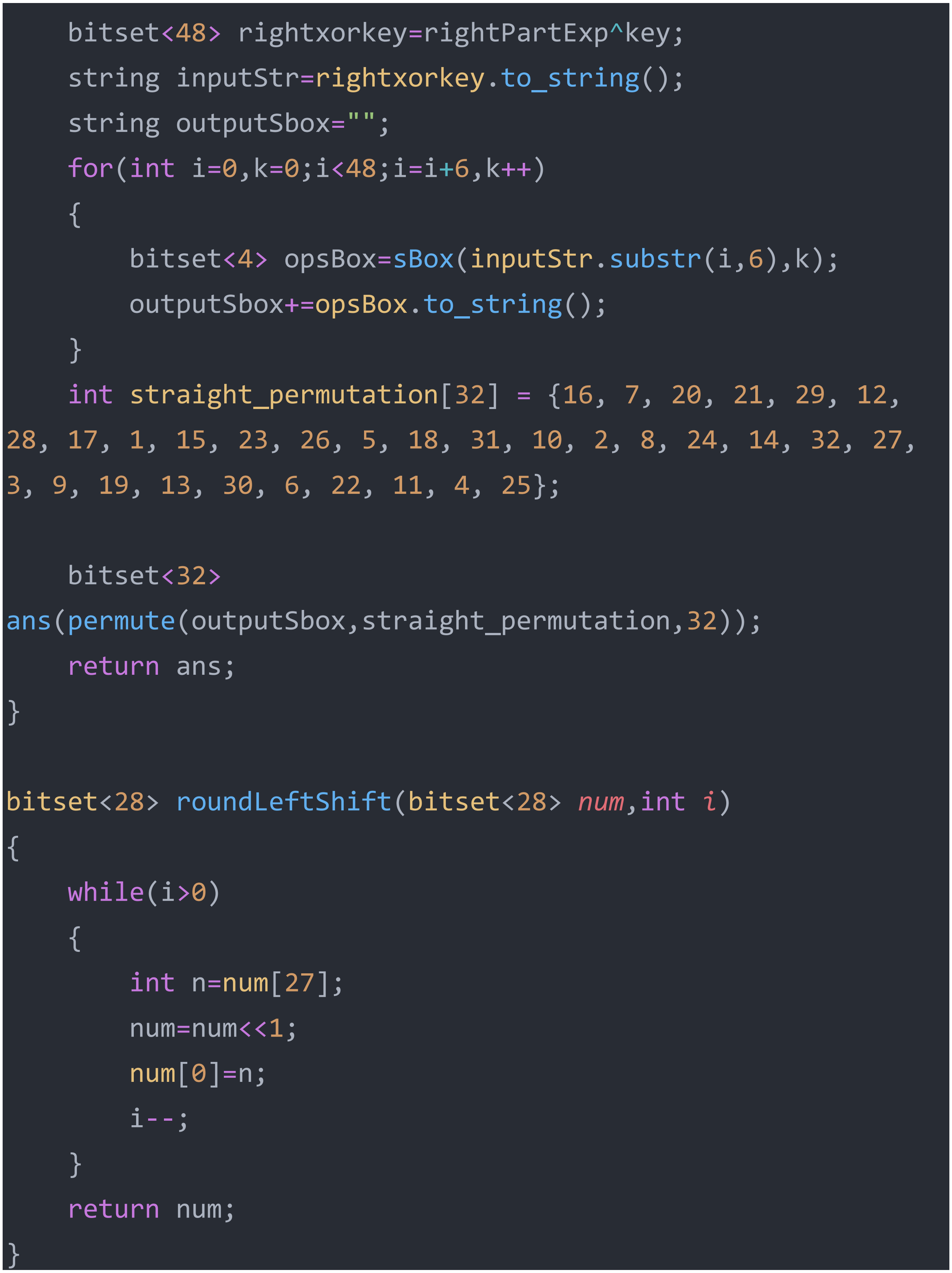
```
#include<bits/stdc++.h>
using namespace std;
string hex2bin(string s)
{
    // hexadecimal to binary conversion
    unordered_map<char, string> mp;
    mp['0'] = "0000";
    mp['1'] = "0001";
    mp['2'] = "0010";
    mp['3'] = "0011";
    mp['4'] = "0100";
    mp['5'] = "0101";
    mp['6'] = "0110";
    mp['7'] = "0111";
```



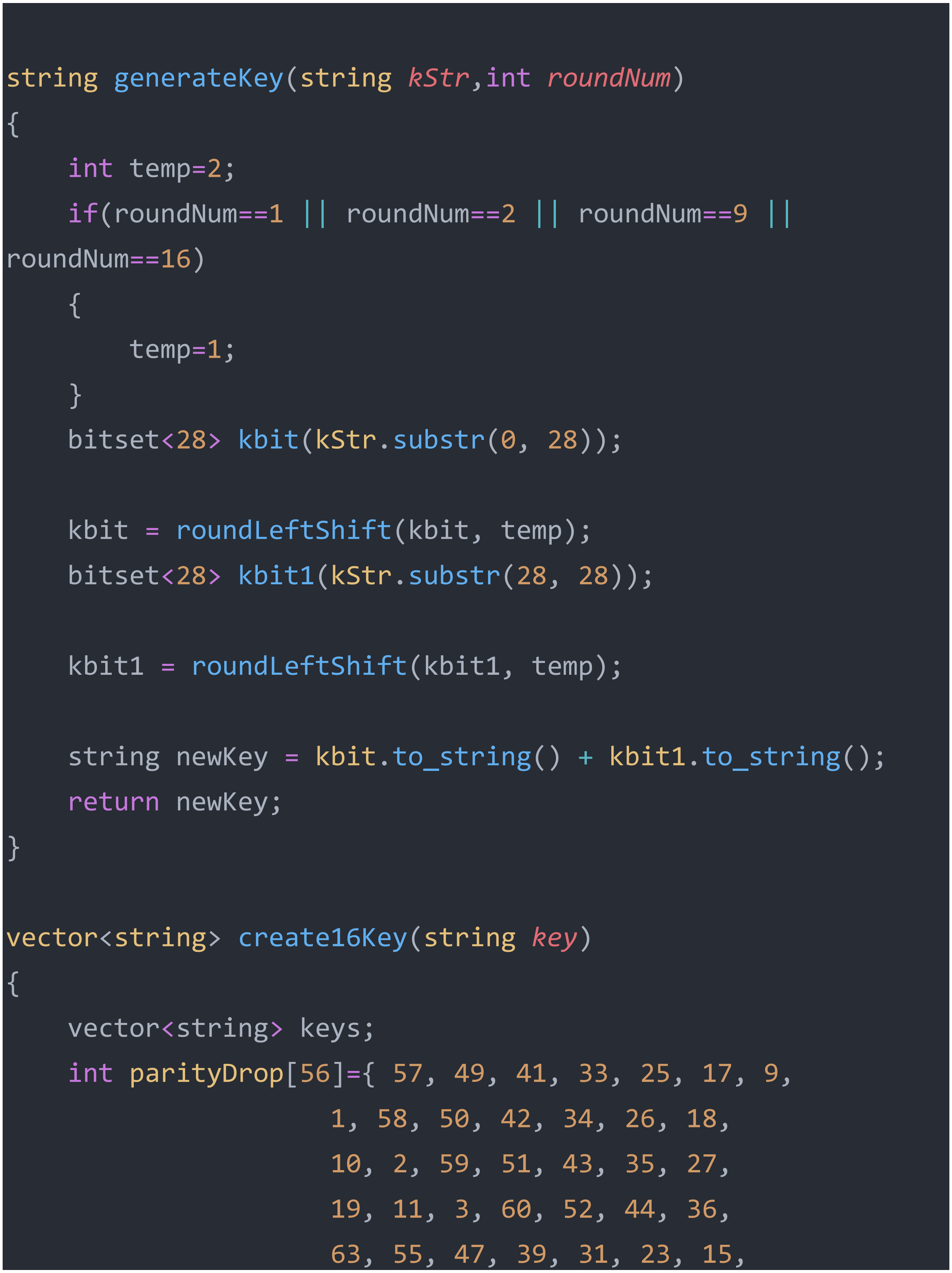
```
    mp['8'] = "1000";
    mp['9'] = "1001";
    mp['A'] = "1010";
    mp['B'] = "1011";
    mp['C'] = "1100";
    mp['D'] = "1101";
    mp['E'] = "1110";
    mp['F'] = "1111";
    string bin = "";
    for (int i = 0; i < s.size(); i++) {
        bin += mp[s[i]];
    }
    return bin;
}
string bin2hex(string s)
{
    // binary to hexadecimal conversion
    unordered_map<string, string> mp;
    mp["0000"] = "0";
    mp["0001"] = "1";
    mp["0010"] = "2";
    mp["0011"] = "3";
    mp["0100"] = "4";
    mp["0101"] = "5";
    mp["0110"] = "6";
    mp["0111"] = "7";
    mp["1000"] = "8";
    mp["1001"] = "9";
```



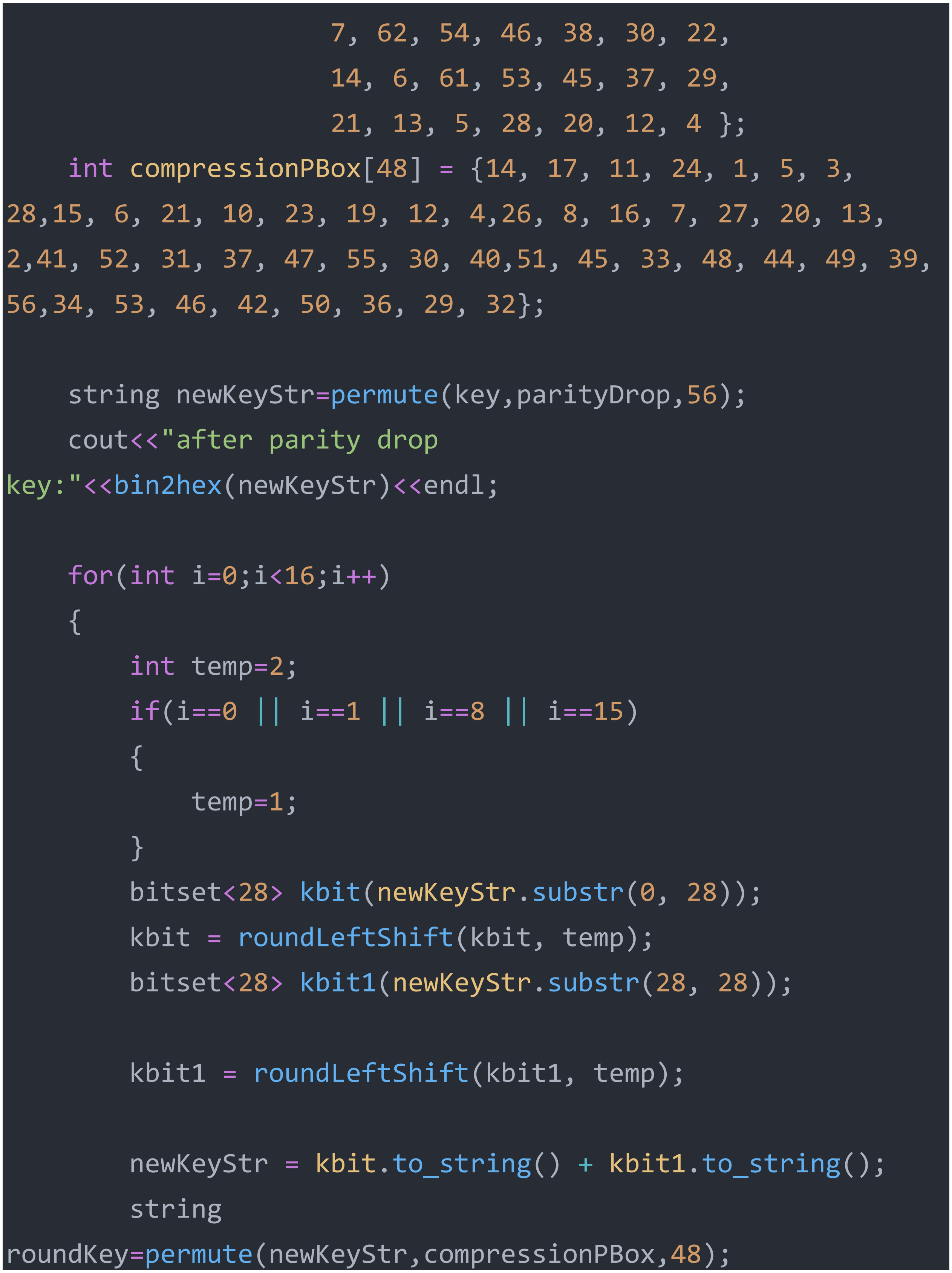
```
    mp["1010"] = "A";
    mp["1011"] = "B";
    mp["1100"] = "C";
    mp["1101"] = "D";
    mp["1110"] = "E";
    mp["1111"] = "F";
    string hex = "";
    for (int i = 0; i < s.length(); i += 4) {
        string ch = "";
        ch += s[i];
        ch += s[i + 1];
        ch += s[i + 2];
        ch += s[i + 3];
        hex += mp[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 inputStr,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]={inputStr[0],inputStr[5],'\0'};
    int row=stoi(rowBit,0,2);
    char
colBit[5]={inputStr[1],inputStr[2],inputStr[3],inputStr[4],'
\0'};
    int col= stoi(colBit, 0, 2);
    bitset<4> res=sbox[num][row][col];
    return res;
}

bitset<32> roundFun(bitset<32> rightPart,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(rightPart.to_string(),pBoxExpansion,48);
    bitset<48> rightPartExp(rightStr);
```



```
    bitset<48> rightxorkey=rightPartExp^key;
    string inputStr=rightxorkey.to_string();
    string outputSbox="";
    for(int i=0,k=0;i<48;i=i+6,k++)
    {
        bitset<4> opsBox=sBox(inputStr.substr(i,6),k);
        outputSbox+=opsBox.to_string();
    }
    int straight_permutation[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,straight_permutation,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;
}
```

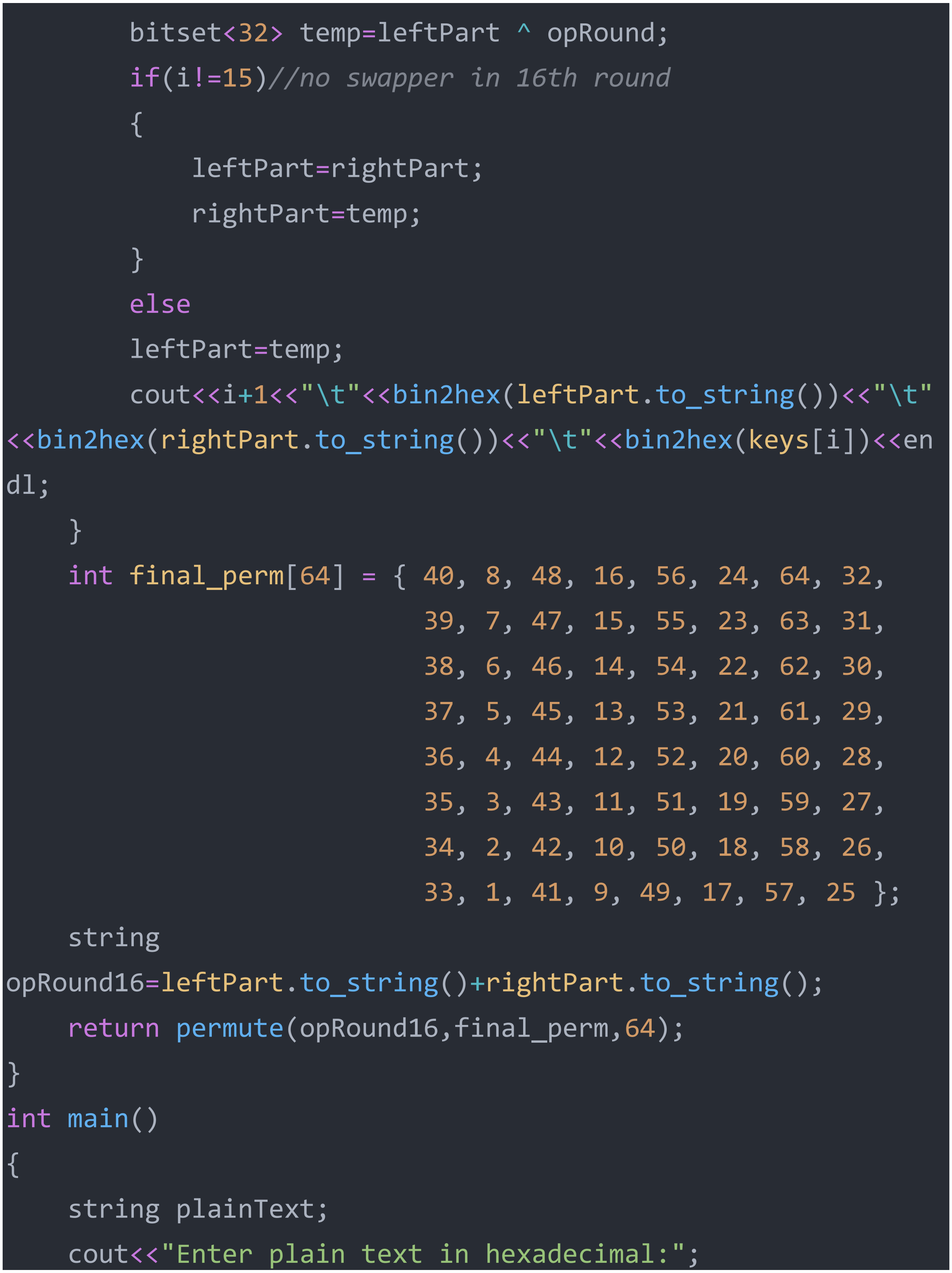


```
string generateKey(string kStr,int roundNum)
{
    int temp=2;
    if(roundNum==1 || roundNum==2 || roundNum==9 ||
roundNum==16)
    {
        temp=1;
    }
    bitset<28> kbit(kStr.substr(0, 28));

    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> create16Key(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,
```

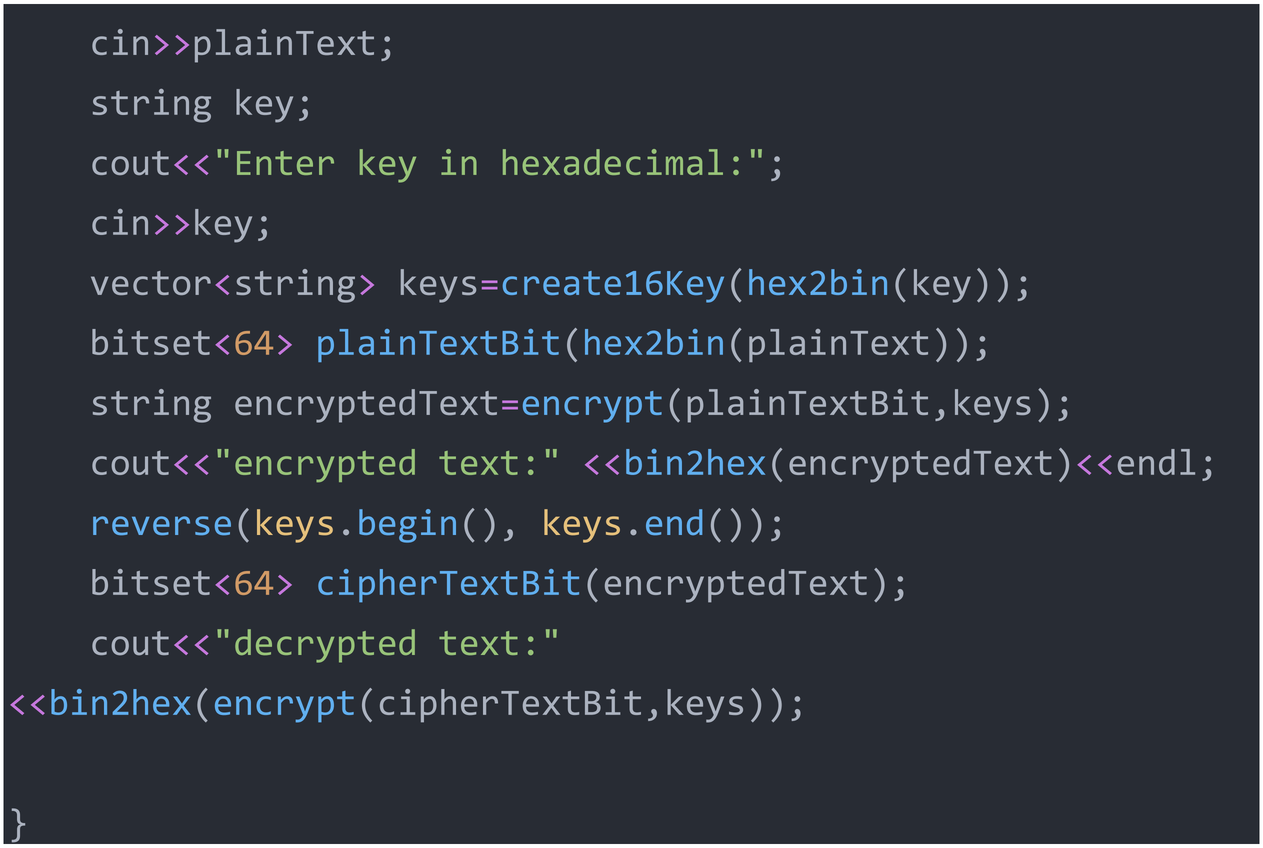


```
                     7, 62, 54, 46, 38, 30, 22,
                     14, 6, 61, 53, 45, 37, 29,
                     21, 13, 5, 28, 20, 12, 4 };
    int compressionPBox[48] = {14, 17, 11, 24, 1, 5, 3,
28,15, 6, 21, 10, 23, 19, 12, 4,26, 8, 16, 7, 27, 20, 13,
2,41, 52, 31, 37, 47, 55, 30, 40,51, 45, 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;
```

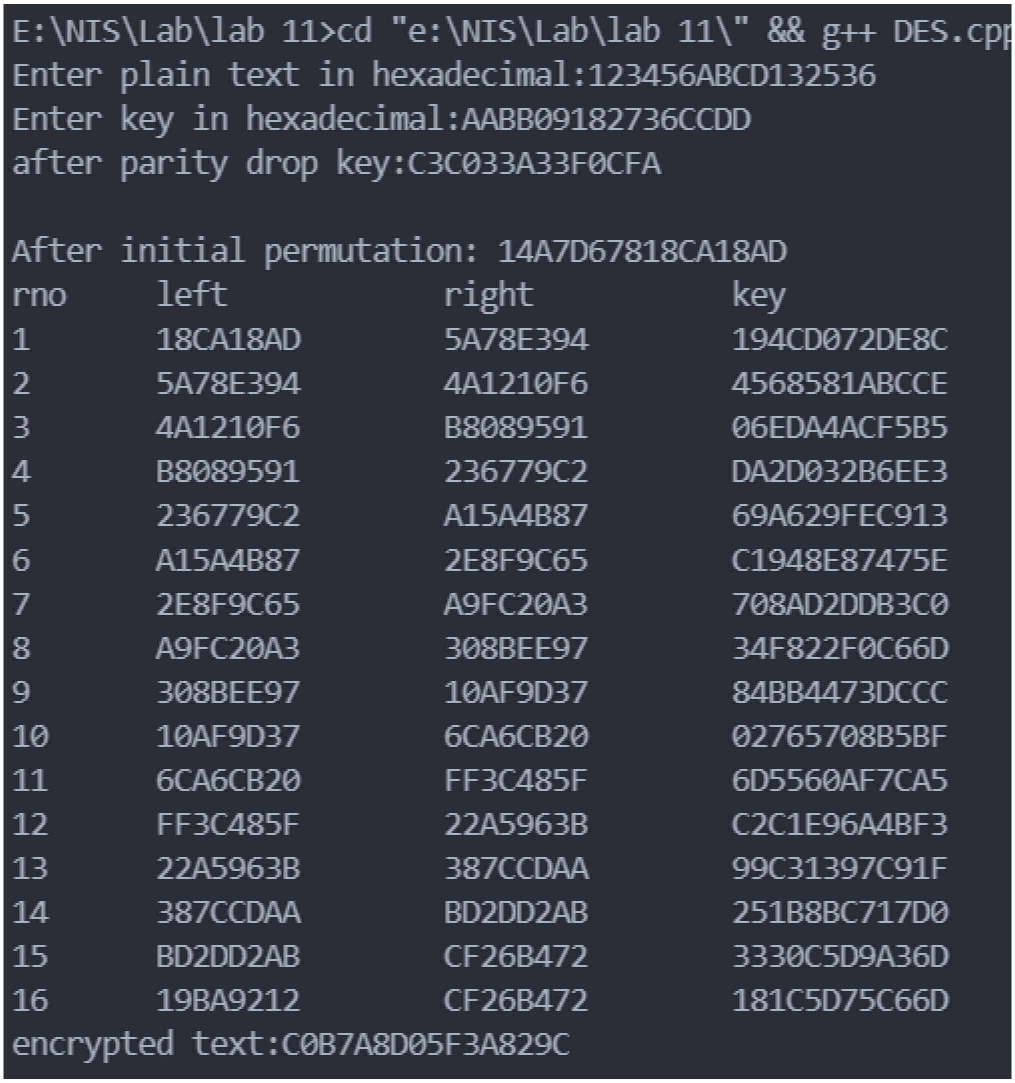


```
        bitset<32> temp=leftPart ^ opRound;
        if(i!=15)//no swapper in 16th round
        {
            leftPart=rightPart;
            rightPart=temp;
        }
        else
        leftPart=temp;
        cout<<i+1<<"\t"<<bin2hex(leftPart.to_string())<<"\t"
<<bin2hex(rightPart.to_string())<<"\t"<<bin2hex(keys[i])<<en
dl;
    }
    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()
{
    string plainText;
    cout<<"Enter plain text in hexadecimal:";
```



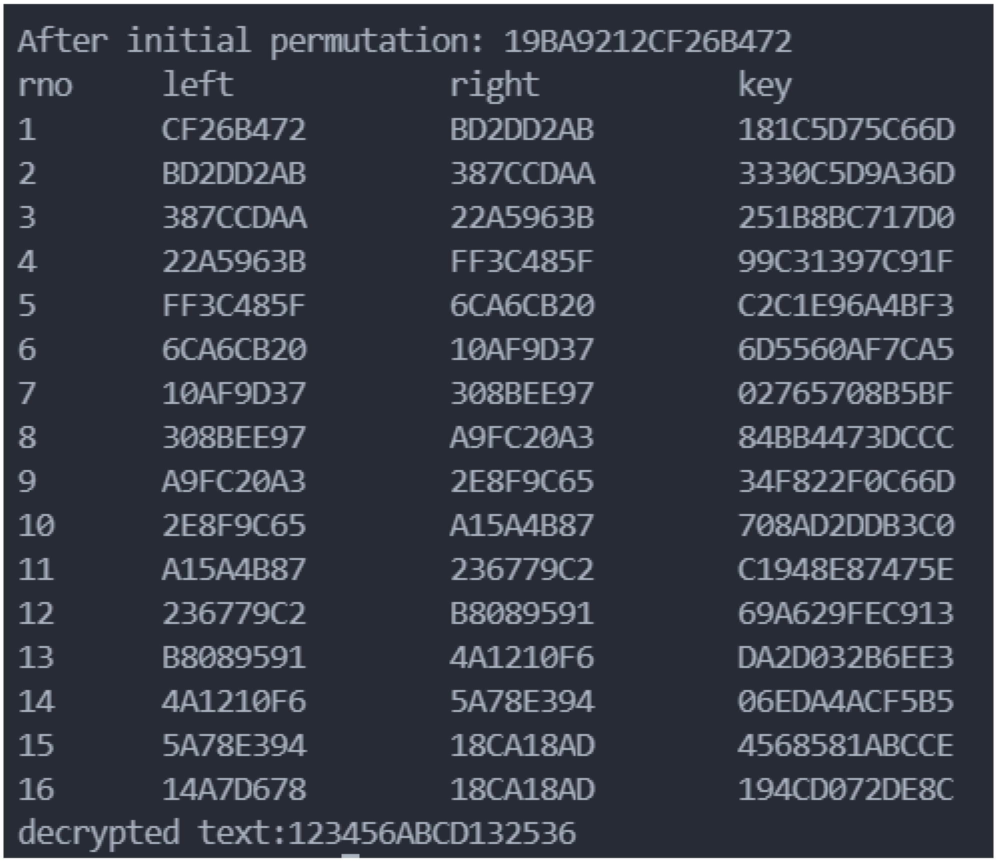
```
    cin>>plainText;
    string key;
    cout<<"Enter key in hexadecimal:";
    cin>>key;
    vector<string> keys=create16Key(hex2bin(key));
    bitset<64> plainTextBit(hex2bin(plainText));
    string encryptedText=encrypt(plainTextBit,keys);
    cout<<"encrypted text:" <<bin2hex(encryptedText)<<endl;
    reverse(keys.begin(), keys.end());
    bitset<64> cipherTextBit(encryptedText);
    cout<<"decrypted text:"
<<bin2hex(encrypt(cipherTextBit,keys));

}
```

Encryption:



```
E:\NIS\Lab\lab 11>cd "e:\NIS\Lab\lab 11\" && g++ DES.cpp
Enter plain text in hexadecimal:123456ABCD132536
Enter key in hexadecimal:AABB09182736CCDD
after parity drop key:C3C033A33F0CFA

After initial permutation: 14A7D67818CA18AD
rno     left            right           key
1       18CA18AD        5A78E394        194CD072DE8C
2       5A78E394        4A1210F6        4568581ABCCE
3       4A1210F6        B8089591        06EDA4ACF5B5
4       B8089591        236779C2        DA2D032B6EE3
5       236779C2        A15A4B87        69A629FEC913
6       A15A4B87        2E8F9C65        C1948E87475E
7       2E8F9C65        A9FC20A3        708AD2DDB3C0
8       A9FC20A3        308BEE97        34F822F0C66D
9       308BEE97        10AF9D37        84BB4473DCCC
10      10AF9D37        6CA6CB20        02765708B5BF
11      6CA6CB20        FF3C485F        6D5560AF7CA5
12      FF3C485F        22A5963B        C2C1E96A4BF3
13      22A5963B        387CCDAA        99C31397C91F
14      387CCDAA        BD2DD2AB        251B8BC717D0
15      BD2DD2AB        CF26B472        3330C5D9A36D
16      19BA9212        CF26B472        181C5D75C66D
encrypted text:C0B7A8D05F3A829C
```

Decryption:



```
After initial permutation: 19BA9212CF26B472
rno     left            right           key
1       CF26B472        BD2DD2AB        181C5D75C66D
2       BD2DD2AB        387CCDAA        3330C5D9A36D
3       387CCDAA        22A5963B        251B8BC717D0
4       22A5963B        FF3C485F        99C31397C91F
5       FF3C485F        6CA6CB20        C2C1E96A4BF3
6       6CA6CB20        10AF9D37        6D5560AF7CA5
7       10AF9D37        308BEE97        02765708B5BF
8       308BEE97        A9FC20A3        84BB4473DCCC
9       A9FC20A3        2E8F9C65        34F822F0C66D
10      2E8F9C65        A15A4B87        708AD2DDB3C0
11      A15A4B87        236779C2        C1948E87475E
12      236779C2        B8089591        69A629FEC913
13      B8089591        4A1210F6        DA2D032B6EE3
14      4A1210F6        5A78E394        06EDA4ACF5B5
15      5A78E394        18CA18AD        4568581ABCCE
16      14A7D678        18CA18AD        194CD072DE8C
decrypted text:123456ABCD132536
```