DES 算法的程序设计和实现报告

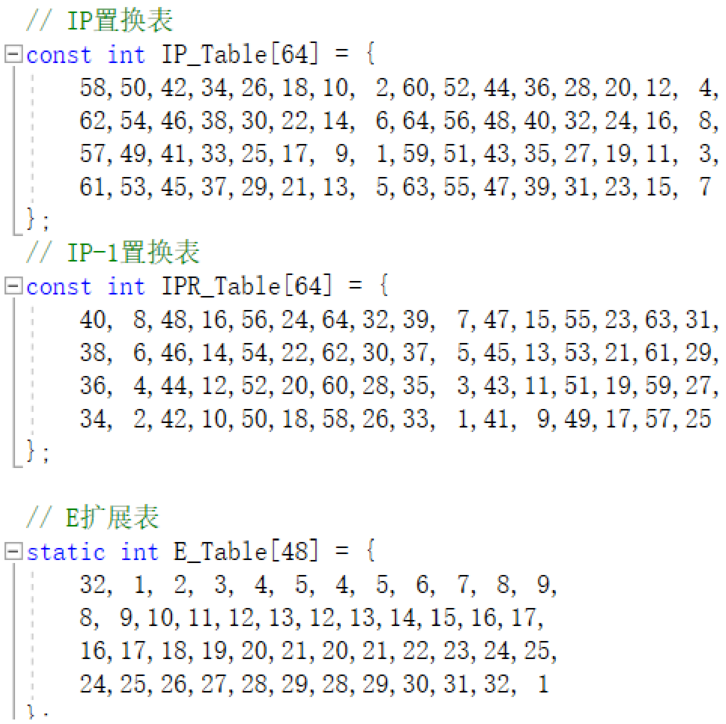
17343128 幸赟

1. 算法原理概述

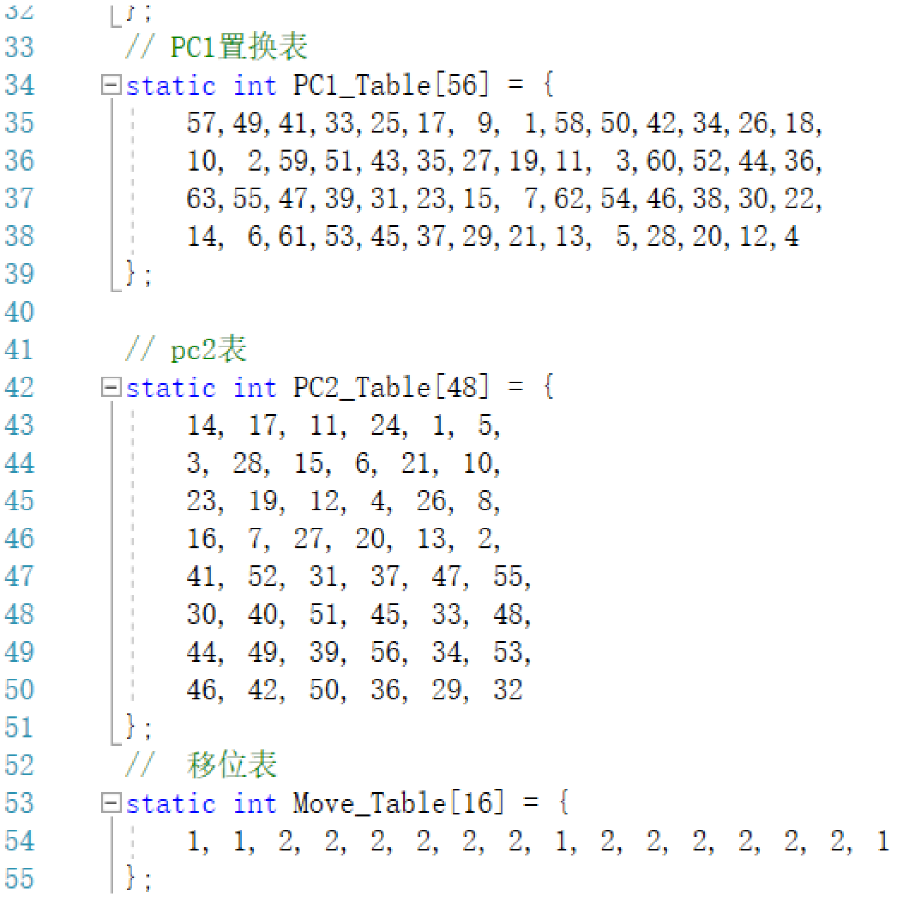
Des 算法是通过一个密钥,将明码和密钥结合进行一系列变换后，生成一个加密后的码，再通过相同的密钥，输入加密后的码，进行解码，便可以得到明码。

2. 总体结构与模块分解

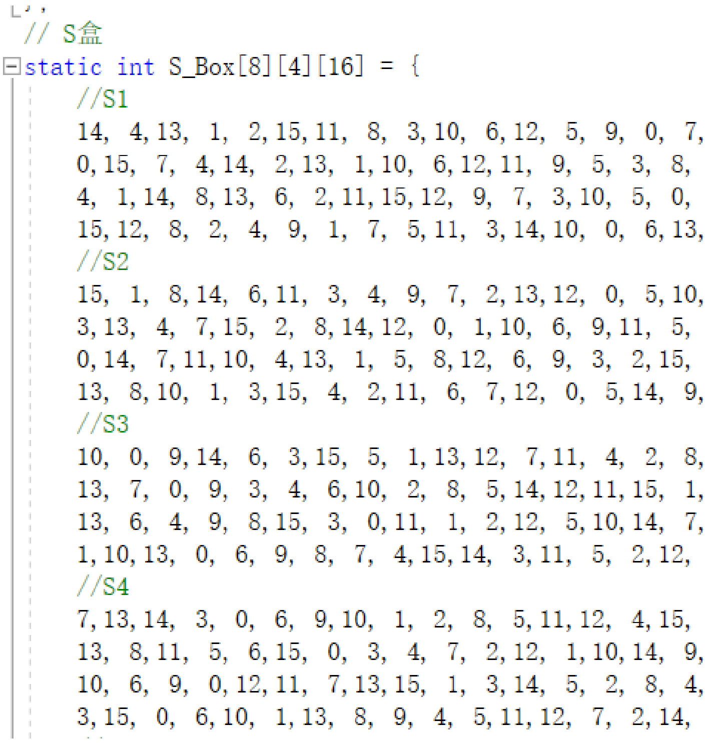
程序总体是分为了 4 个模块，第一个是声明各个 des 算法中需要的各个表，见下图：



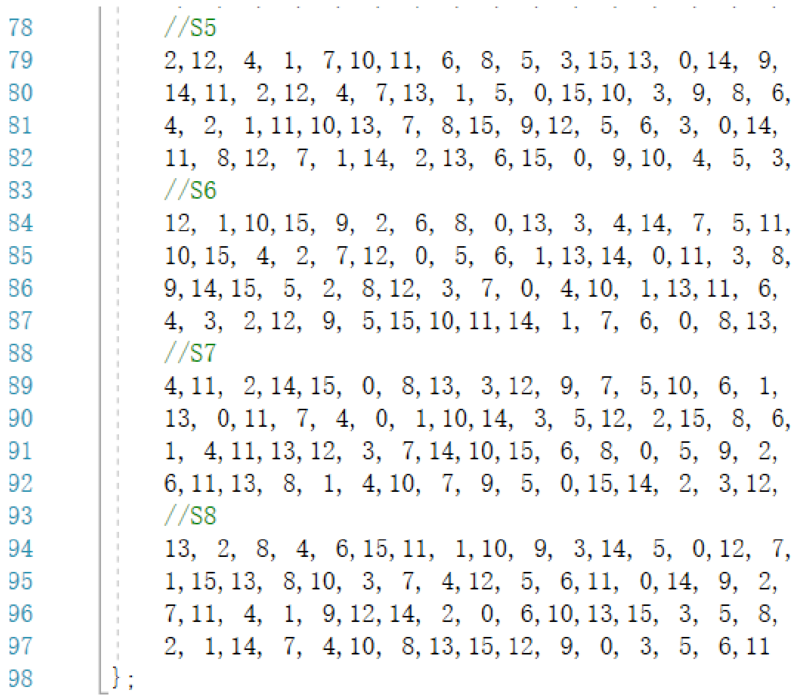
```
// IP置换表
const int IP_Table[64] = {
    58, 50, 42, 34, 26, 18, 10,  2, 60, 52, 44, 36, 28, 20, 12,  4,
    62, 54, 46, 38, 30, 22, 14,  6, 64, 56, 48, 40, 32, 24, 16,  8,
    57, 49, 41, 33, 25, 17,  9,  1, 59, 51, 43, 35, 27, 19, 11,  3,
    61, 53, 45, 37, 29, 21, 13,  5, 63, 55, 47, 39, 31, 23, 15,  7
};
// IP-1置换表
const int IPR_Table[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
};

// E扩展表
static int E_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
};
```



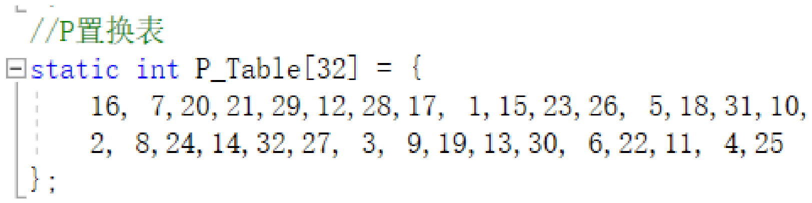
```
    };
    // PC1置换表
    static int PC1_Table[56] = {
        57,49,41,33,25,17, 9, 1,58,50,42,34,26,18,
        10, 2,59,51,43,35,27,19,11, 3,60,52,44,36,
        63,55,47,39,31,23,15, 7,62,54,46,38,30,22,
        14, 6,61,53,45,37,29,21,13, 5,28,20,12,4
    };

    // pc2表
    static int PC2_Table[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
    };
    //  移位表
    static int Move_Table[16] = {
        1,  1,  2,  2,  2,  2,  2,  2,  1,  2,  2,  2,  2,  2,  2,  1
    };
```



```
};
// S盒
static int S_Box[8][4][16] = {
    //S1
    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,
    //S2
    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,
    //S3
    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,
    //S4
    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,
```

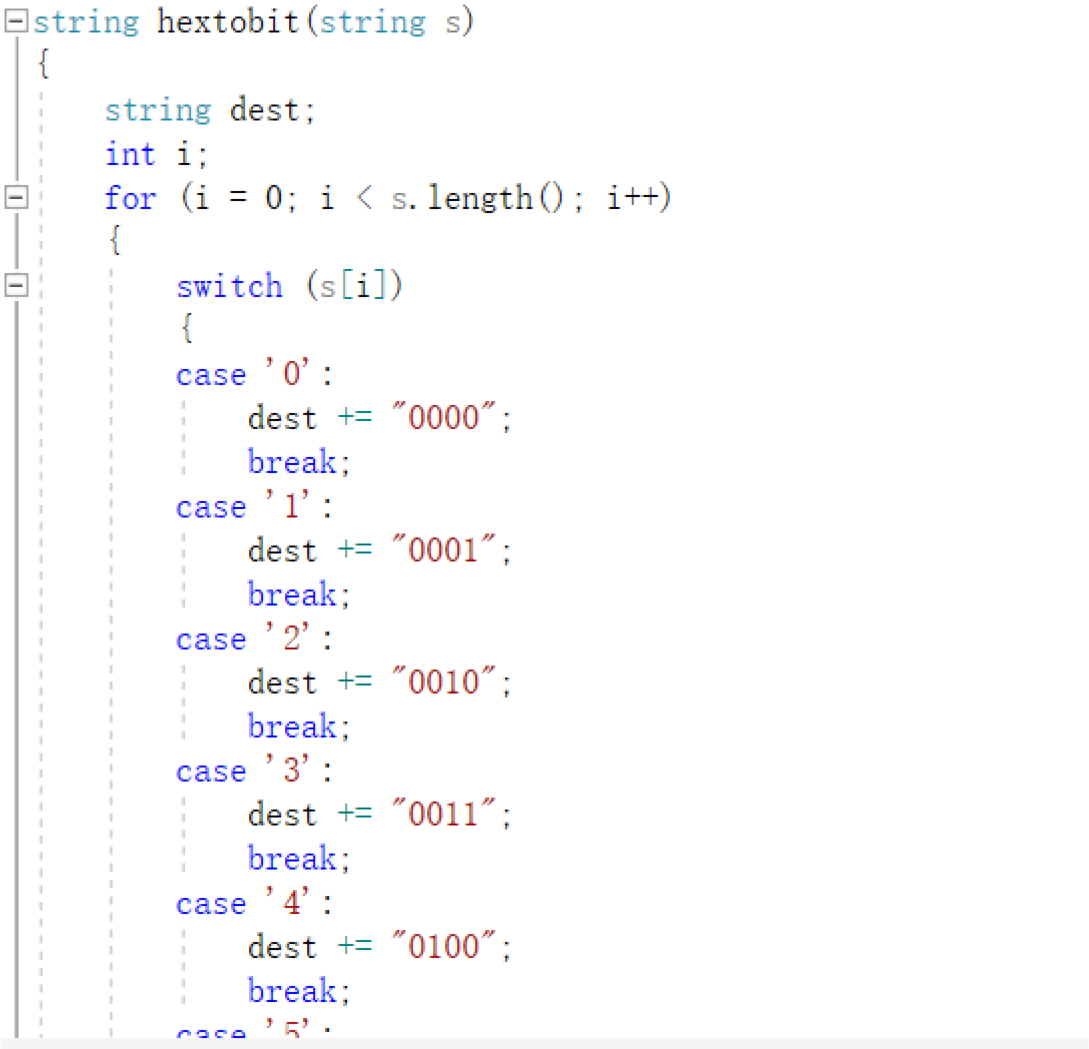


```
    //S5
    2,12, 4, 1, 7,10,11, 6, 8, 5, 3,15,13, 0,14, 9,
    14,11, 2,12, 4, 7,13, 1, 5, 0,15,10, 3, 9, 8, 6,
    4, 2, 1,11,10,13, 7, 8,15, 9,12, 5, 6, 3, 0,14,
    11, 8,12, 7, 1,14, 2,13, 6,15, 0, 9,10, 4, 5, 3,
    //S6
    12, 1,10,15, 9, 2, 6, 8, 0,13, 3, 4,14, 7, 5,11,
    10,15, 4, 2, 7,12, 0, 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,
    //S7
    4,11, 2,14,15, 0, 8,13, 3,12, 9, 7, 5,10, 6, 1,
    13, 0,11, 7, 4, 0, 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,
    //S8
    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
};
```

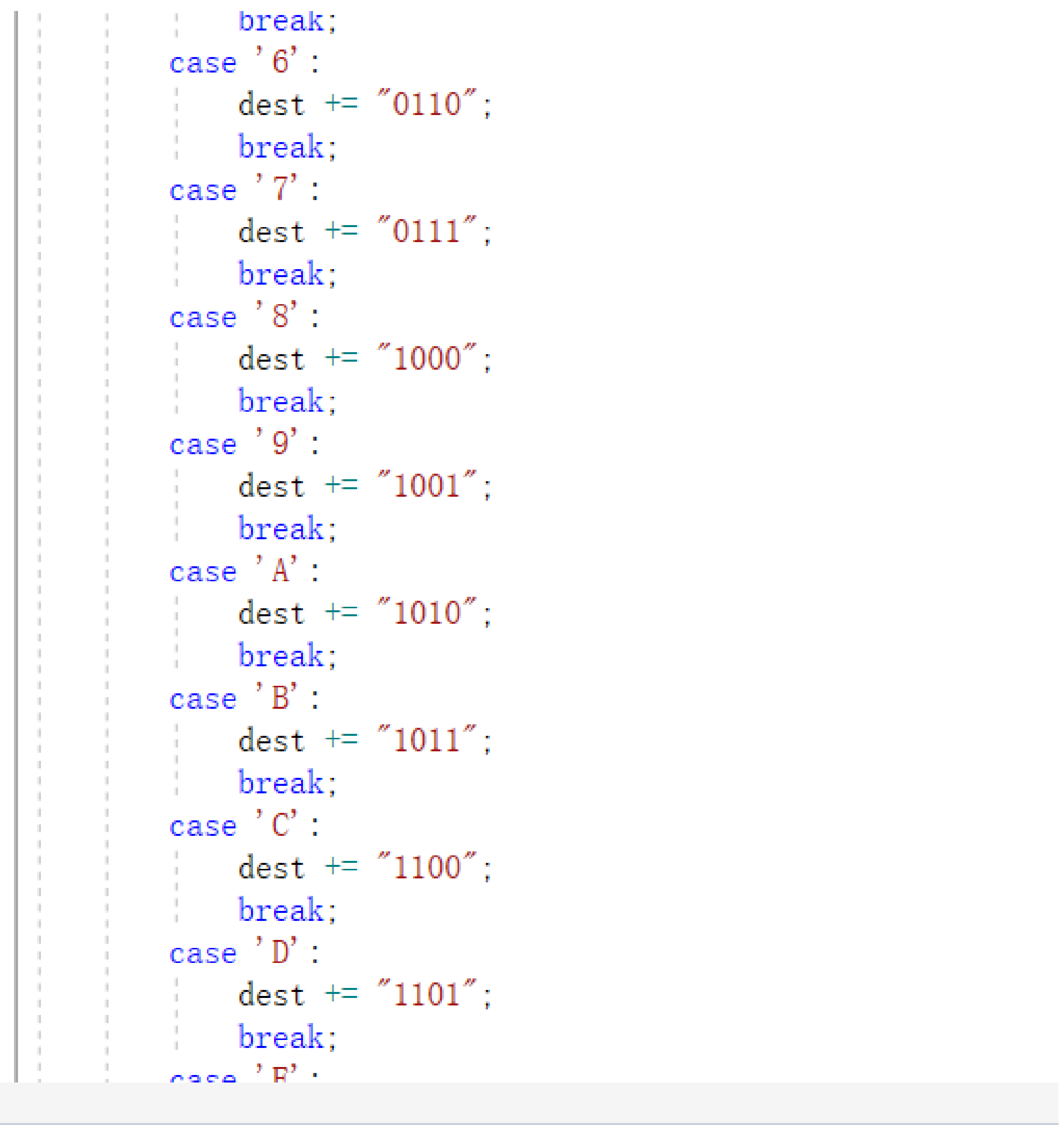


```
//P置换表
static int P_Table[32] = {
    16, 7,20,21,29,12,28,17, 1,15,23,26, 5,18,31,10,
    2, 8,24,14,32,27, 3, 9,19,13,30, 6,22,11, 4,25
};
```

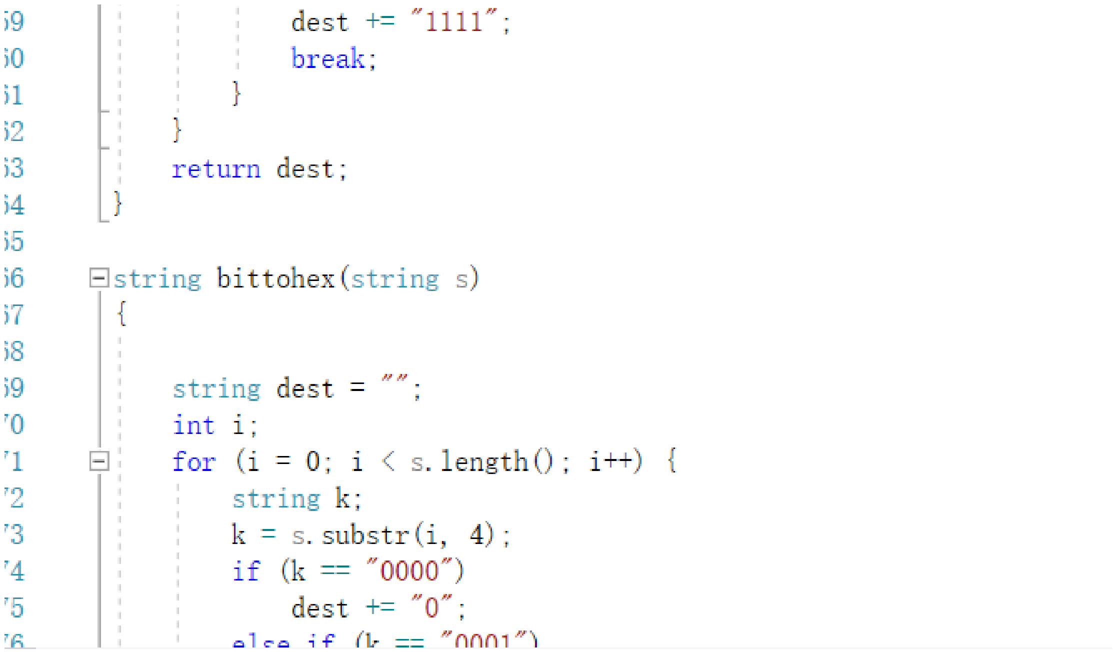
第二个模块是写的进制转换函数，一共有 3 个，分别是十六进制转二进制，二进制转十六进制，十进制转二进制，见下图：



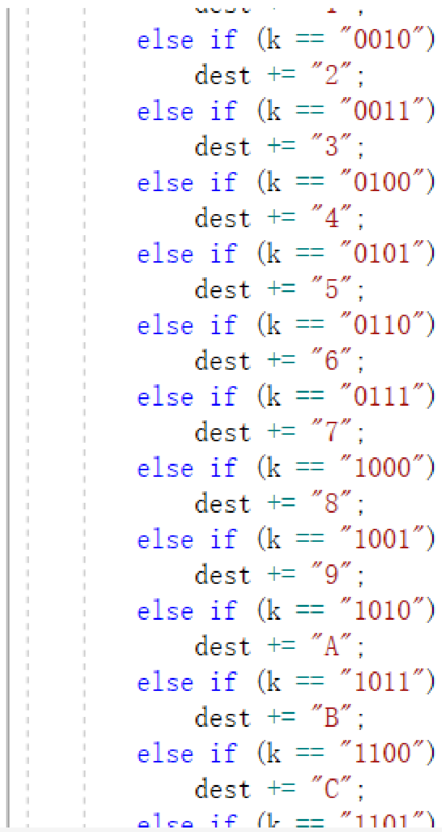
```cpp
string hextobit(string s)
{
    string dest;
    int i;
    for (i = 0; i < s.length(); i++)
    {
        switch (s[i])
        {
        case '0':
            dest += "0000";
            break;
        case '1':
            dest += "0001";
            break;
        case '2':
            dest += "0010";
            break;
        case '3':
            dest += "0011";
            break;
        case '4':
            dest += "0100";
            break;
        case '5':
```



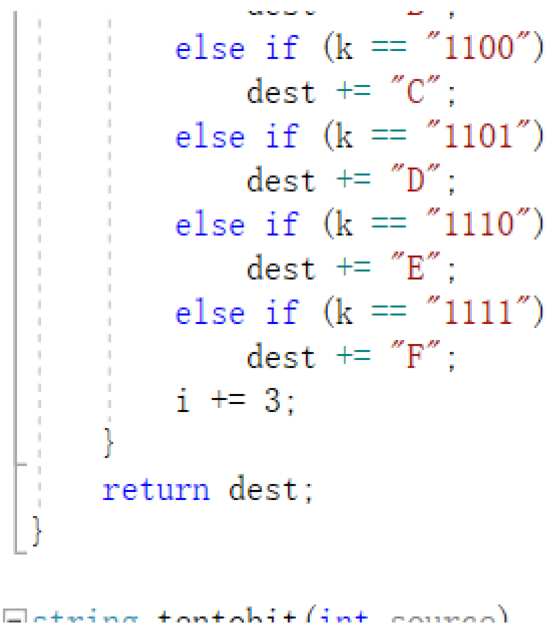
```
                break;
            case '6':
                dest += "0110";
                break;
            case '7':
                dest += "0111";
                break;
            case '8':
                dest += "1000";
                break;
            case '9':
                dest += "1001";
                break;
            case 'A':
                dest += "1010";
                break;
            case 'B':
                dest += "1011";
                break;
            case 'C':
                dest += "1100";
                break;
            case 'D':
                dest += "1101";
                break;
            case 'F':
```



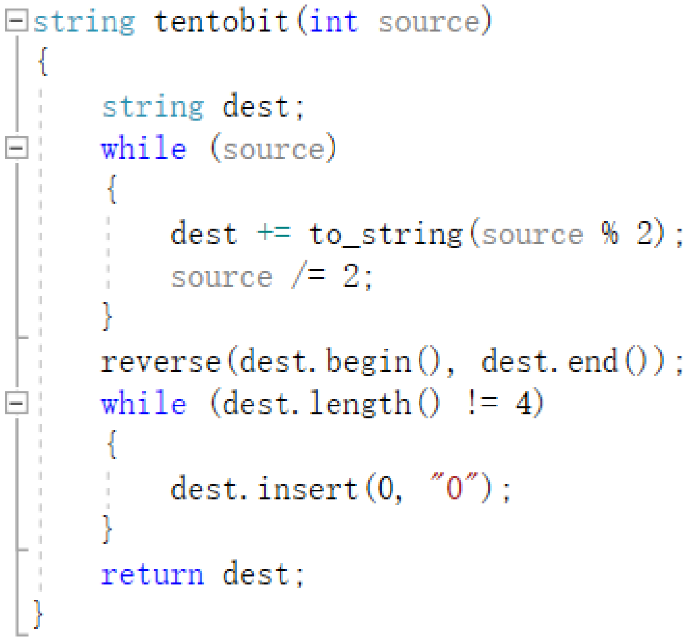
```
                dest += "1111";
                break;
            }
        }
    return dest;
}

string bittohex(string s)
{

    string dest = "";
    int i;
    for (i = 0; i < s.length(); i++) {
        string k;
        k = s.substr(i, 4);
        if (k == "0000")
            dest += "0";
        else if (k == "0001")
```



```
            dest += "1";
        else if (k == "0010")
            dest += "2";
        else if (k == "0011")
            dest += "3";
        else if (k == "0100")
            dest += "4";
        else if (k == "0101")
            dest += "5";
        else if (k == "0110")
            dest += "6";
        else if (k == "0111")
            dest += "7";
        else if (k == "1000")
            dest += "8";
        else if (k == "1001")
            dest += "9";
        else if (k == "1010")
            dest += "A";
        else if (k == "1011")
            dest += "B";
        else if (k == "1100")
            dest += "C";
        else if (k == "1101")
```

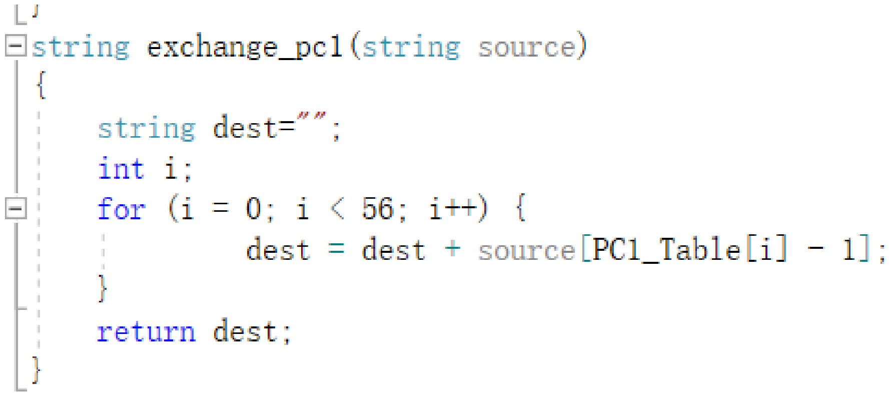


```
            dest += "B";
        else if (k == "1100")
            dest += "C";
        else if (k == "1101")
            dest += "D";
        else if (k == "1110")
            dest += "E";
        else if (k == "1111")
            dest += "F";
        i += 3;
    }
    return dest;
}

string tentobit(int source)
```



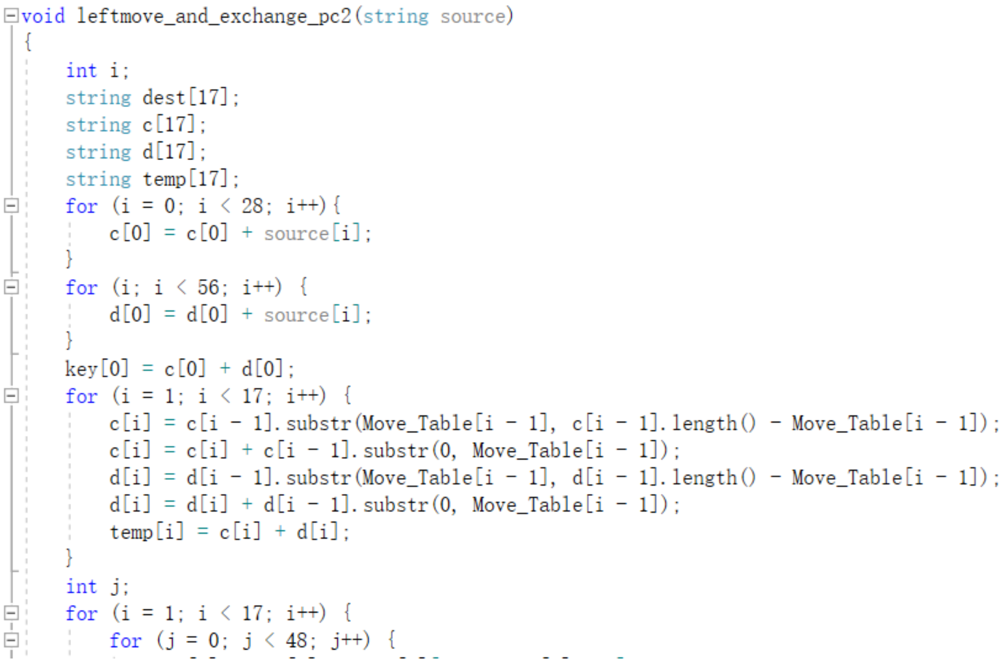
```
string tentobit(int source)
{
    string dest;
    while (source)
    {
        dest += to_string(source % 2);
        source /= 2;
    }
    reverse(dest.begin(), dest.end());
    while (dest.length() != 4)
    {
        dest.insert(0, "0");
    }
    return dest;
}
```

有了进制转换，便开始着手写各个步骤的代码，分别有，pc1 转换

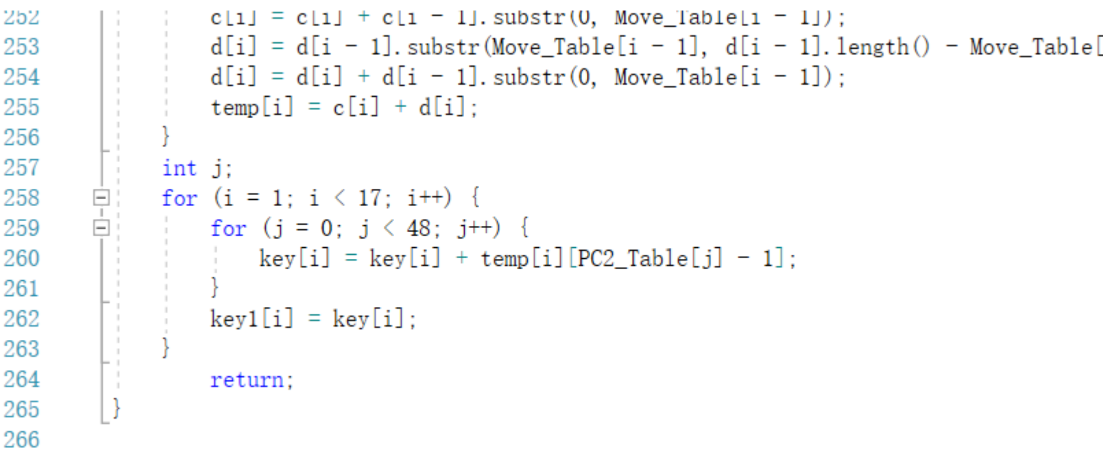


```
string exchange_pc1(string source)
{
    string dest="";
    int i;
    for (i = 0; i < 56; i++) {
            dest = dest + source[PC1_Table[i] - 1];
    }
    return dest;
}
```

左移以及 pc2 转换，得到 16 个 key



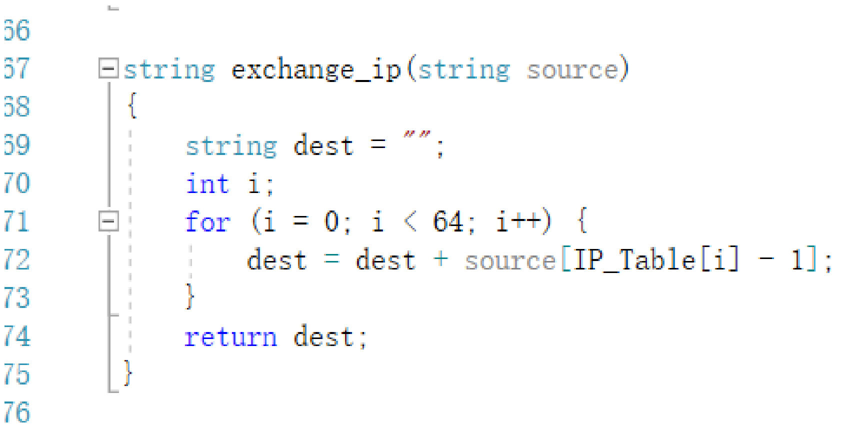
```
void leftmove_and_exchange_pc2(string source)
{
    int i;
    string dest[17];
    string c[17];
    string d[17];
    string temp[17];
    for (i = 0; i < 28; i++){
        c[0] = c[0] + source[i];
    }
    for (i; i < 56; i++) {
        d[0] = d[0] + source[i];
    }
    key[0] = c[0] + d[0];
    for (i = 1; i < 17; i++) {
        c[i] = c[i - 1].substr(Move_Table[i - 1], c[i - 1].length() - Move_Table[i - 1]);
        c[i] = c[i] + c[i - 1].substr(0, Move_Table[i - 1]);
        d[i] = d[i - 1].substr(Move_Table[i - 1], d[i - 1].length() - Move_Table[i - 1]);
        d[i] = d[i] + d[i - 1].substr(0, Move_Table[i - 1]);
        temp[i] = c[i] + d[i];
    }
    int j;
    for (i = 1; i < 17; i++) {
        for (j = 0; j < 48; j++) {
```



```
            c[i] = c[i] + c[i - 1].substr(0, Move_Table[i - 1]);
            d[i] = d[i - 1].substr(Move_Table[i - 1], d[i - 1].length() - Move_Table[
            d[i] = d[i] + d[i - 1].substr(0, Move_Table[i - 1]);
            temp[i] = c[i] + d[i];
        }
        int j;
        for (i = 1; i < 17; i++) {
            for (j = 0; j < 48; j++) {
                key[i] = key[i] + temp[i][PC2_Table[j] - 1];
            }
            key1[i] = key[i];
        }
            return;
    }
```

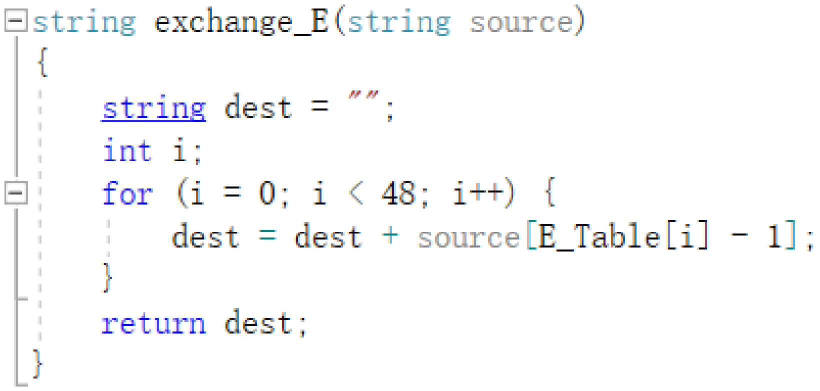
其中 key 是保存在全局变量中以便后面使用

再是 ip 转换



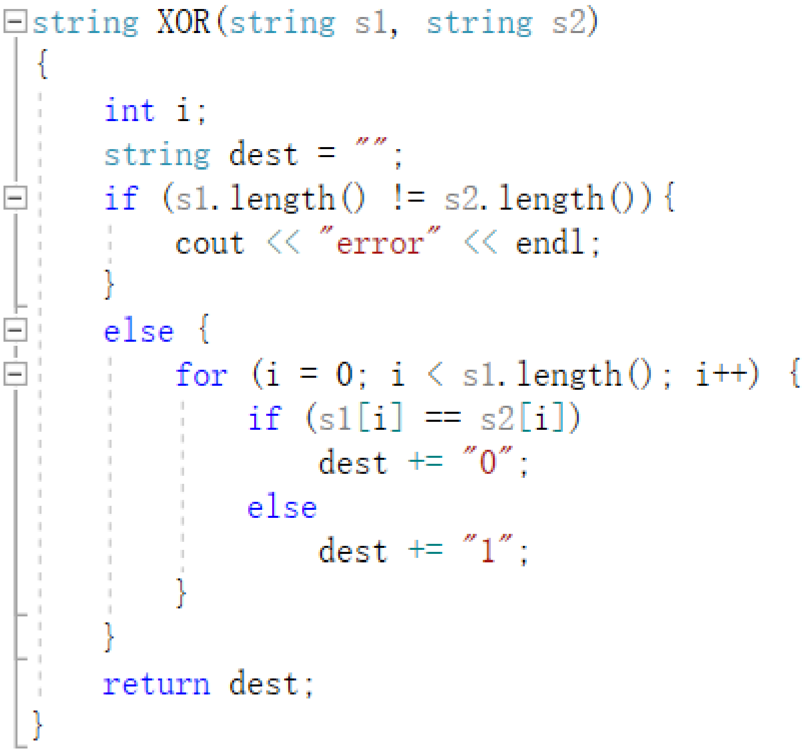
```
string exchange_ip(string source)
{
    string dest = "";
    int i;
    for (i = 0; i < 64; i++) {
        dest = dest + source[IP_Table[i] - 1];
    }
    return dest;
}
```

然后 E 转换



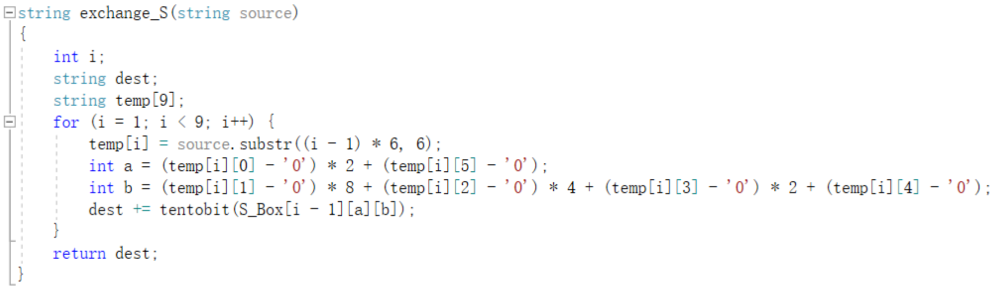
```
string exchange_E(string source)
{
    string dest = "";
    int i;
    for (i = 0; i < 48; i++) {
        dest = dest + source[E_Table[i] - 1];
    }
    return dest;
}
```

异或函数



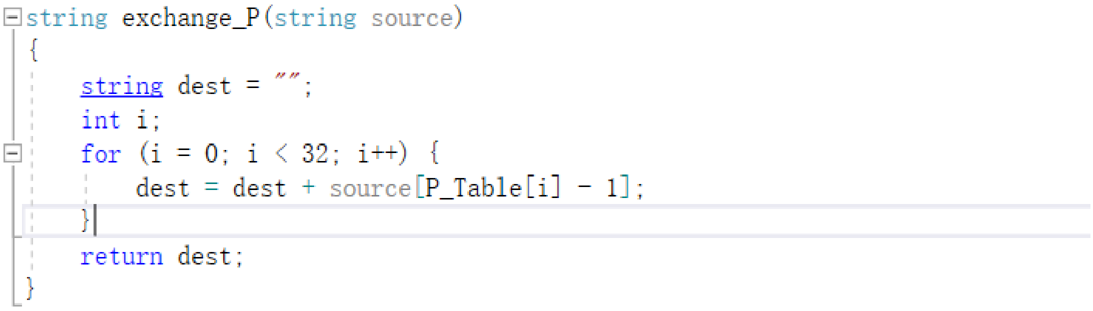
```
string XOR(string s1, string s2)
{
    int i;
    string dest = "";
    if (s1.length() != s2.length()){
        cout << "error" << endl;
    }
    else {
        for (i = 0; i < s1.length(); i++) {
            if (s1[i] == s2[i])
                dest += "0";
            else
                dest += "1";
        }
    }
    return dest;
}
```

再是 S 盒转换



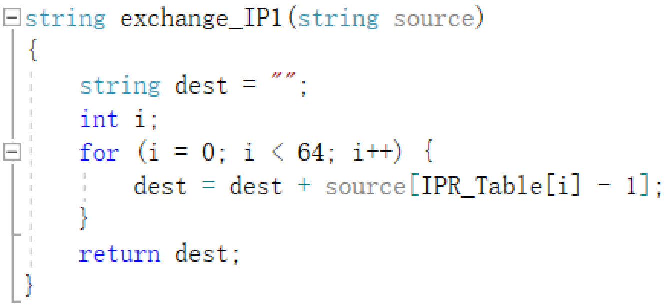
```
string exchange_S(string source)
{
    int i;
    string dest;
    string temp[9];
    for (i = 1; i < 9; i++) {
        temp[i] = source.substr((i - 1) * 6, 6);
        int a = (temp[i][0] - '0') * 2 + (temp[i][5] - '0');
        int b = (temp[i][1] - '0') * 8 + (temp[i][2] - '0') * 4 + (temp[i][3] - '0') * 2 + (temp[i][4] - '0');
        dest += tentobit(S_Box[i - 1][a][b]);
    }
    return dest;
}
```

P 转换



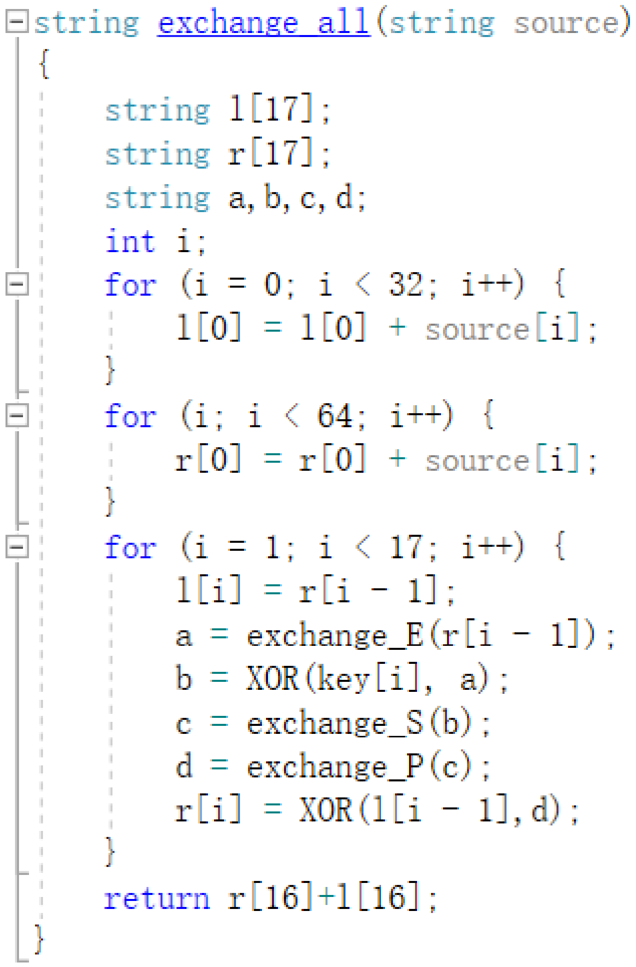
```
string exchange_P(string source)
{
    string dest = "";
    int i;
    for (i = 0; i < 32; i++) {
        dest = dest + source[P_Table[i] - 1];
    }
    return dest;
}
```

IP1 转换



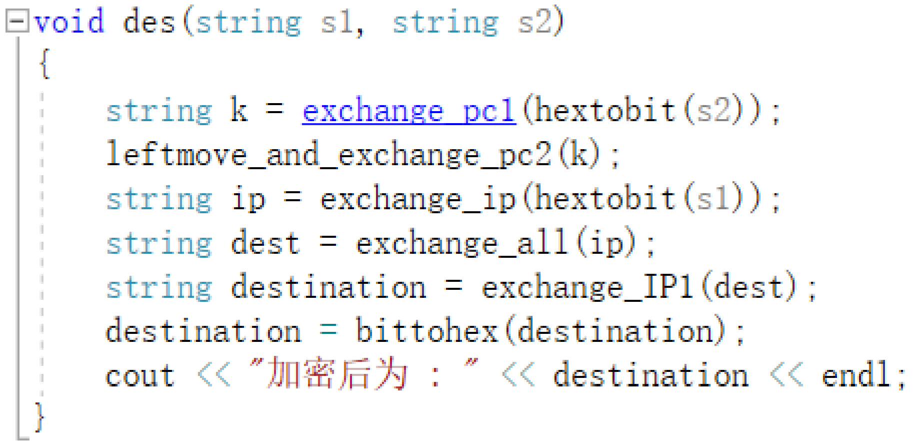
```
string exchange_IP1(string source)
{
    string dest = "";
    int i;
    for (i = 0; i < 64; i++) {
        dest = dest + source[IPR_Table[i] - 1];
    }
    return dest;
}
```

再是将以上函数联立起来，得到 IP，即加密后的码



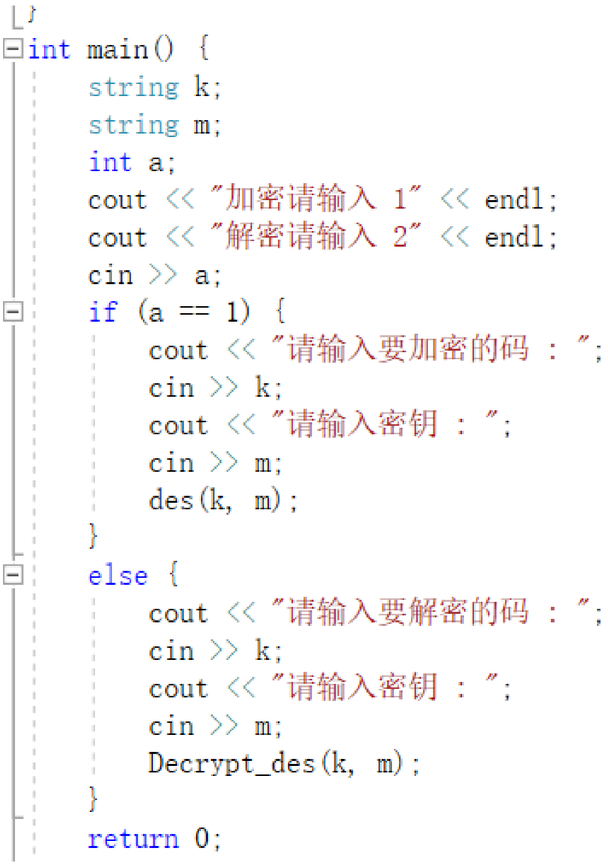
```
string exchange_all(string source)
{
    string l[17];
    string r[17];
    string a,b,c,d;
    int i;
    for (i = 0; i < 32; i++) {
        l[0] = l[0] + source[i];
    }
    for (i; i < 64; i++) {
        r[0] = r[0] + source[i];
    }
    for (i = 1; i < 17; i++) {
        l[i] = r[i - 1];
        a = exchange_E(r[i - 1]);
        b = XOR(key[i], a);
        c = exchange_S(b);
        d = exchange_P(c);
        r[i] = XOR(l[i - 1],d);
    }
    return r[16]+l[16];
}
```

最后是 des 解码，调用其中的函数完成



```
void des(string s1, string s2)
{
    string k = exchange_pc1(hextobit(s2));
    leftmove_and_exchange_pc2(k);
    string ip = exchange_ip(hextobit(s1));
    string dest = exchange_all(ip);
    string destination = exchange_IP1(dest);
    destination = bittohex(destination);
    cout << "加密后为 : " << destination << endl;
}
```

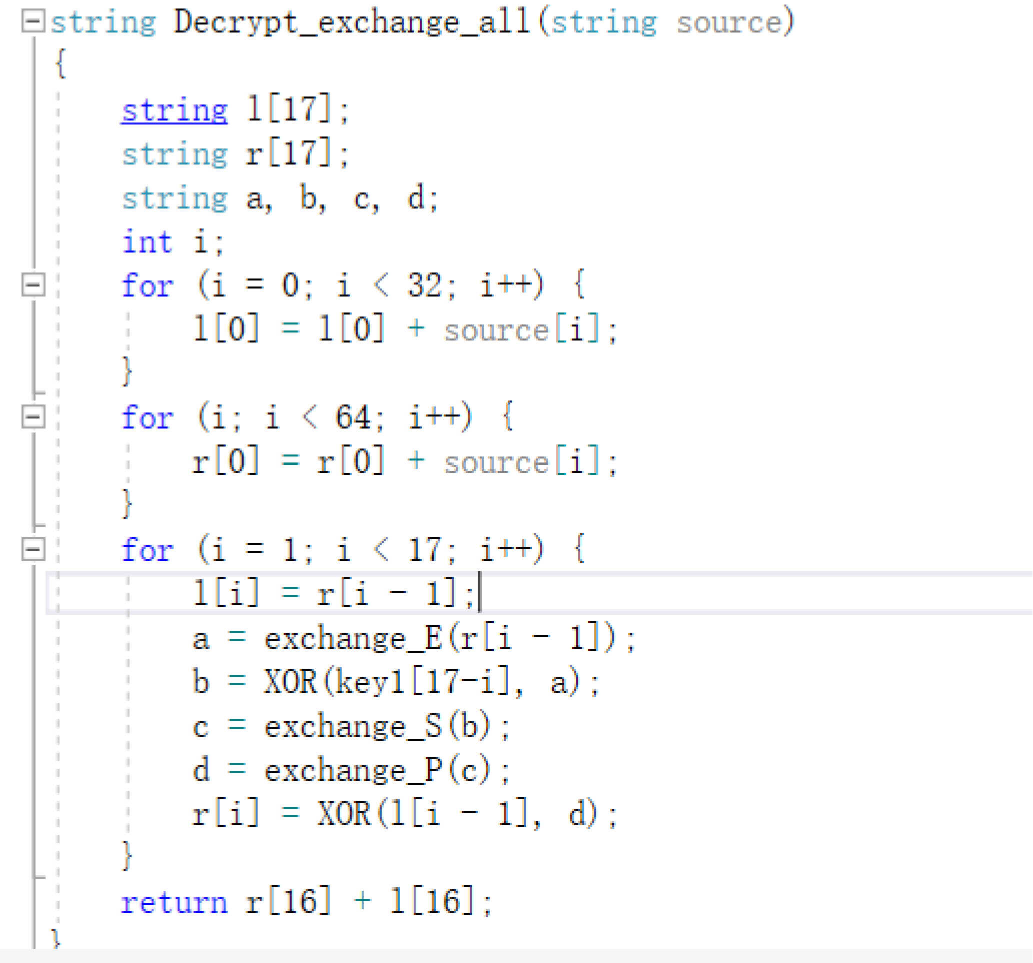
然后第四个模块是 main 函数，主要是一些 ui 问题，



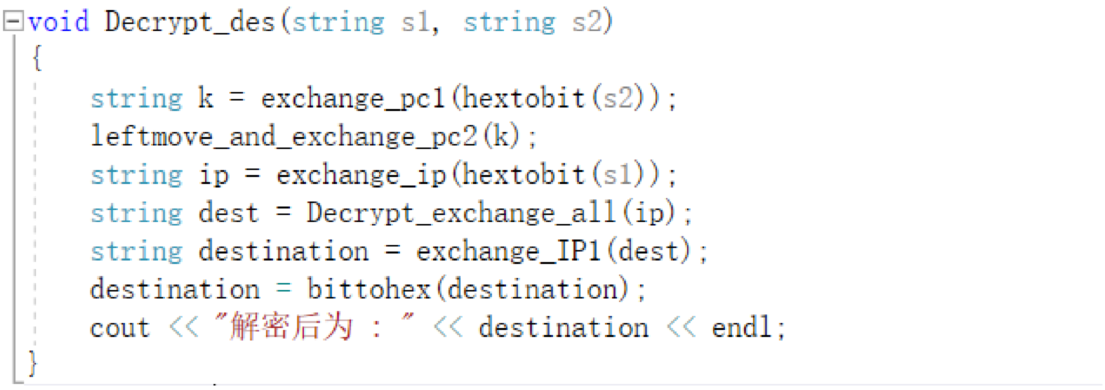
```
}
int main() {
    string k;
    string m;
    int a;
    cout << "加密请输入 1" << endl;
    cout << "解密请输入 2" << endl;
    cin >> a;
    if (a == 1) {
        cout << "请输入要加密的码 : ";
        cin >> k;
        cout << "请输入密钥 : ";
        cin >> m;
        des(k, m);
    }
    else {
        cout << "请输入要解密的码 : ";
        cin >> k;
        cout << "请输入密钥 : ";
        cin >> m;
        Decrypt_des(k, m);
    }
    return 0;
```

其中解码基本上和加密相同，只需要重写两个函数，一个是exchange_all，另一个是整个的 des

而修改的只需要把 key 异或的时候，从 1-16 改为 16-1 即可，见下图：



```
string Decrypt_exchange_all(string source)
{
    string l[17];
    string r[17];
    string a, b, c, d;
    int i;
    for (i = 0; i < 32; i++) {
        l[0] = l[0] + source[i];
    }
    for (i; i < 64; i++) {
        r[0] = r[0] + source[i];
    }
    for (i = 1; i < 17; i++) {
        l[i] = r[i - 1];
        a = exchange_E(r[i - 1]);
        b = XOR(key1[17-i], a);
        c = exchange_S(b);
        d = exchange_P(c);
        r[i] = XOR(l[i - 1], d);
    }
    return r[16] + l[16];
}
```



```
void Decrypt_des(string s1, string s2)
{
    string k = exchange_pc1(hextobit(s2));
    leftmove_and_exchange_pc2(k);
    string ip = exchange_ip(hextobit(s1));
    string dest = Decrypt_exchange_all(ip);
    string destination = exchange_IP1(dest);
    destination = bittohex(destination);
    cout << "解密后为 : " << destination << endl;
}
```

以上便是整个程序的全部内容。

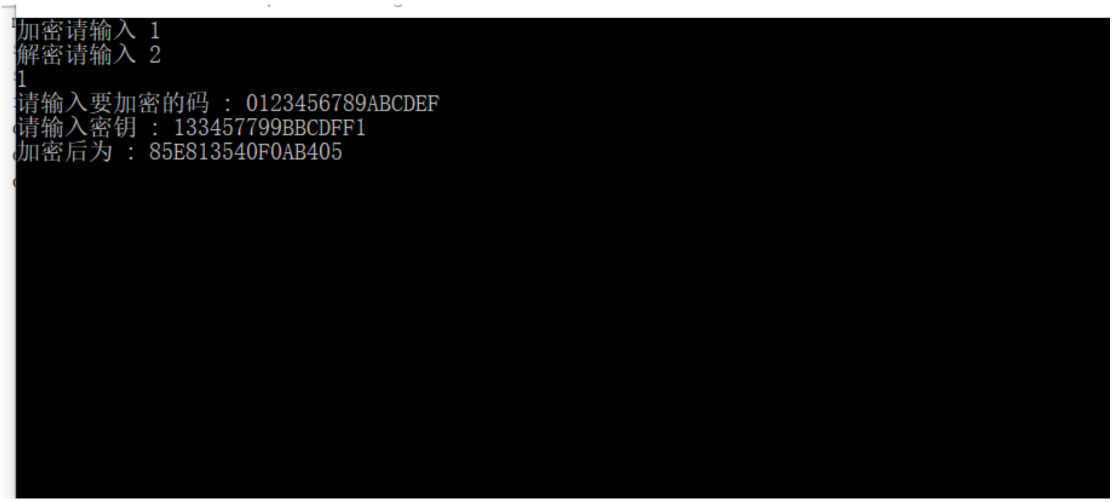
3. 数据结构

本次实验基本没用到什么数据结构，只用到了 string 以及一些与 string 相关的函数。

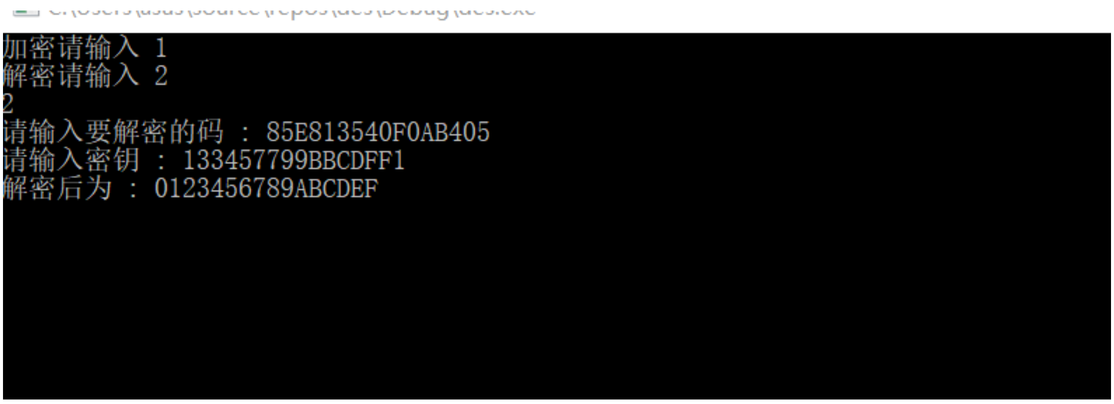
4. c 语言源代码

见压缩包中另外的文件

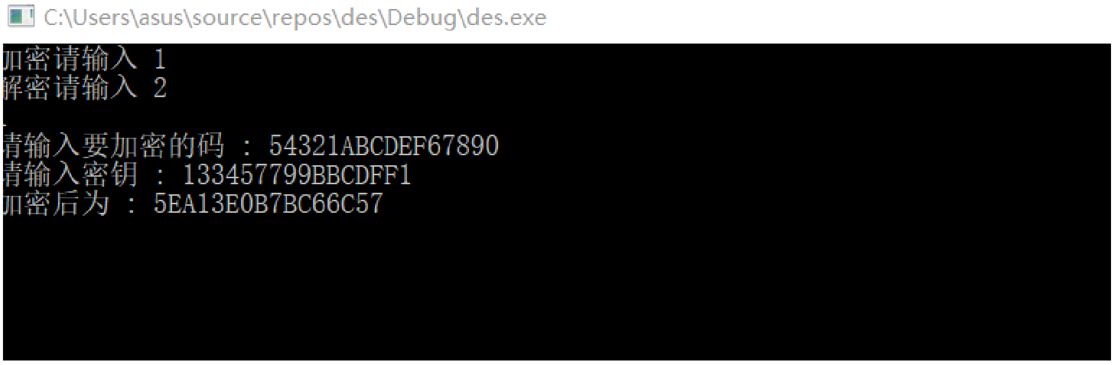
5. 实验截图



```
加密请输入 1
解密请输入 2
1
请输入要加密的码 : 0123456789ABCDEF
请输入密钥 : 133457799BBCDFF1
加密后为 : 85E813540F0AB405
```

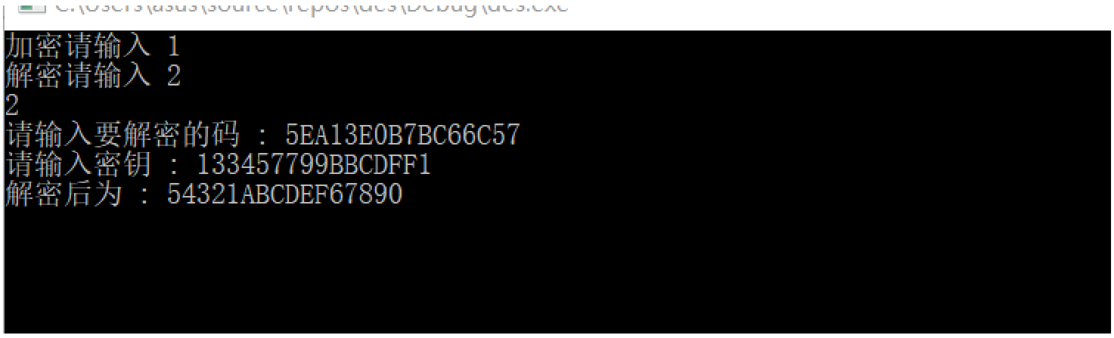


```
加密请输入 1
解密请输入 2
2
请输入要解密的码 : 85E813540F0AB405
请输入密钥 : 133457799BBCDFF1
解密后为 : 0123456789ABCDEF
```



C:\Users\asus\source\repos\des\Debug\des.exe

```
加密请输入 1
解密请输入 2

请输入要加密的码 : 54321ABCDEF67890
请输入密钥 : 133457799BBCDFF1
加密后为 : 5EA13E0B7BC66C57
```



C:\Users\asus\source\repos\des\Debug\des.exe

```
加密请输入 1
解密请输入 2
2
请输入要解密的码 : 5EA13E0B7BC66C57
请输入密钥 : 133457799BBCDFF1
解密后为 : 54321ABCDEF67890
```