密码学实验一　DES 加密算法

08111505 1120152056 李世林

目录

密码学实验一　DES 加密算法

一、算法原理

1、DES 加密

DES 加密主要由四个部分完成：

1、初始置换 IP；

2、获取每轮的加密子密钥 Ki；

3、使用函数 f 进行 16 轮加密；

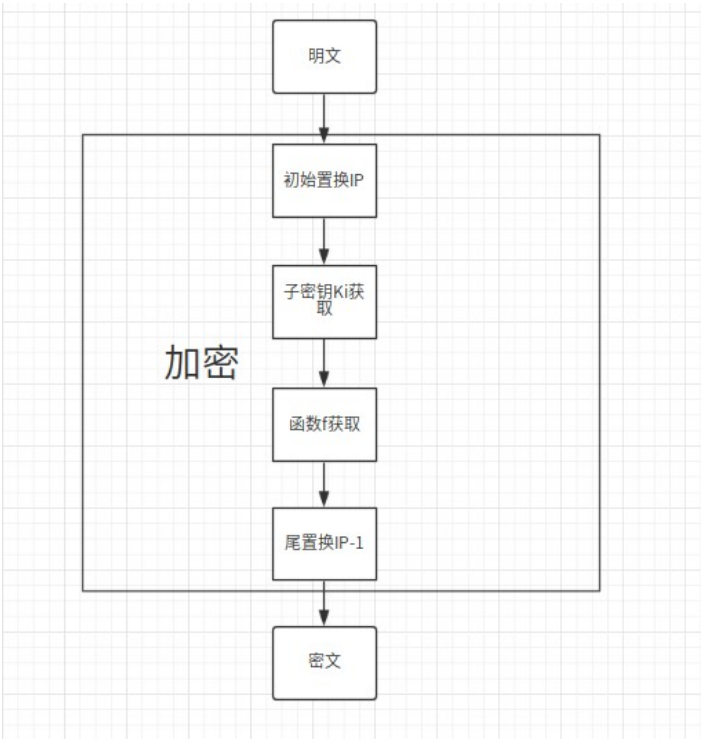
4、尾置换 IP-1 ；

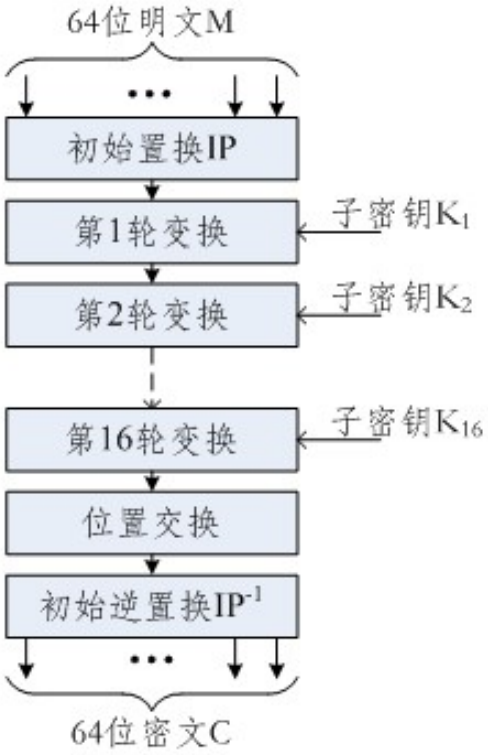
2、DES 解密

DES 解密算法与加密算法完全相同，只需要将密文作为输入，秘钥使用加密时使用的密钥，注意使用子密钥时将使用顺序反过来。

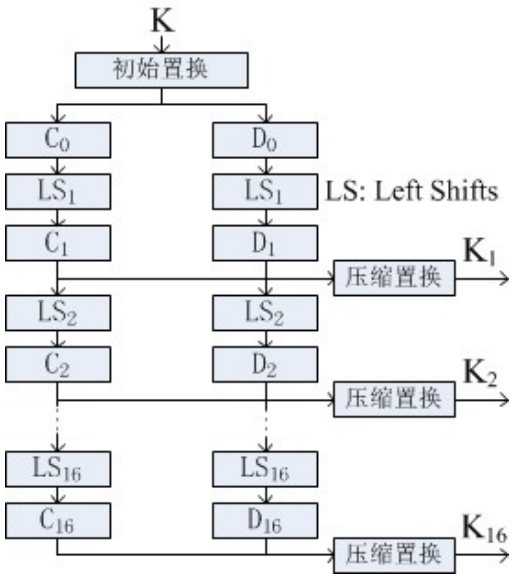
3、加密流程图

加密流程图如下所示：



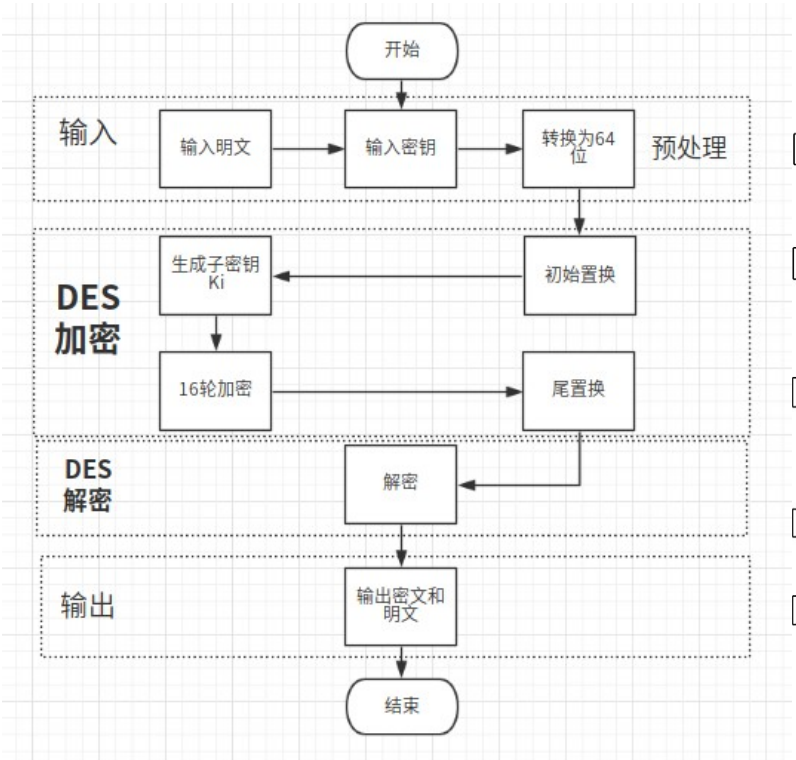


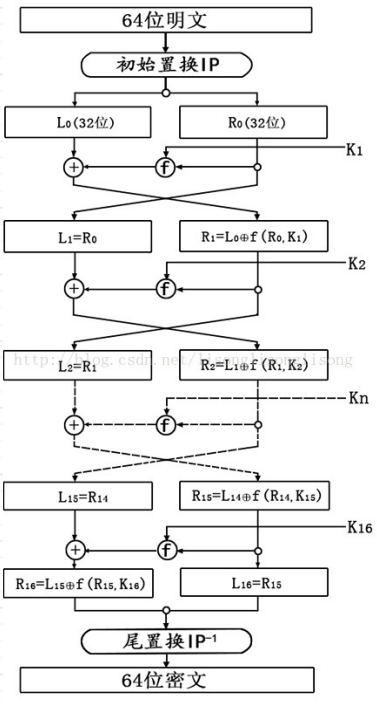
生成 16 轮加密所需的子密钥的过程如下：



二、C++代码实现 DES 加密解密

1、程序执行流程图





2、代码解释

2.1 输入

用户通过标准输入方式输入需要加密的明文 plain 和密钥 K。

程序将用户输入的明文按字节转换成 64 位的待加密的明文，不足 64 位的将使用空白字符填充。

用户输入的加密字符限定数字，英文字母大小写和其他非空白字符，中文字符则输入 4 位。然后将其转换成 64 位的密钥。

转换代码：

```
//将char字符数组转为二进制
bitset<64> charToBitset(const char s[8])
{
    bitset<64> bits;
    for (int i = 0; i<8; ++i)
        for (int j = 0; j<8; ++j)
            bits[i * 8 + j] = ((s[i] >> j) & 1);
    return bits;
}
```

2.2 初始置换

使用的初始置换表如下

```
// 初始置换表
int IP[] = {
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 };
```

根据表中的规定，将输入的 64 位明文重新进行排序，即将第 58 位放到第 1 位，第 50 位放到第 2 位……以此类推。初始置换以后得到的是一个 64 位的输出。

2.3 获取子密钥 Ki

密钥置换表和密钥压缩置换表如下

```
// 密钥置换表，将64位密钥变成56位
int PC_1[] = {
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 };
```

```
// 压缩置换，将56位密钥压缩成48位子密钥
int PC_2[] = {
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 };
```

用户输入 64 位的密钥，根据密钥置换表 PC-1，去掉了奇偶校验位,将 64 位变成 56 位密钥。

循环左移表

```
// 每轮左移的位数
int shiftBits[] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };
```

将 PC-1 置换得到的 56 位密钥，分为前 28 位 C0 和后 28 位 D0，分别对它们进行循环左移，C0 左移得到 C1，D0 左移得到 D1。

```
//对56位密钥的前后部分进行左移
bitset<28> leftShift(bitset<28> k, int shift)
{
    bitset<28> tmp = k;
    for (int i = 27; i >= 0; --i)
    {
        if (i - shift<0)
            k[i] = tmp[i - shift + 28];
        else
            k[i] = tmp[i - shift];
    }
    return k;
}
```

将 C1 和 D1 合并成 56 位,然后通过 PC-2 表进行压缩置换,得到当前这一轮的 48 位子密钥 K1 。

然后对 C1 和 D1 进行左移和压缩置换，获取下一轮的子密钥……一共进行 16 轮，

得到 16 个 48 位的子密钥。

2.4 加密

密码函数 f(R, K)接受两个输入：32 位的数据和 48 位的子密钥。然后：

通过表 E 进行扩展置换，将输入的 32 位数据扩展为 48 位；

扩展置换表如下：

```
// 扩展置换表，将 32位 扩展至 48位
int E[] = {
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 };
```

将扩展后的 48 位数据与 48 位的子密钥进行异或运算；

```
//生成16个48位的子密钥
void generateKeys()
{
    bitset<56> realKey;
    bitset<28> left;
    bitset<28> right;
    bitset<48> compressKey;
    // 去掉奇偶标记位，将64位密钥变成56位
    for (int i = 0; i<56; ++i)
        realKey[55 - i] = key[64 - PC_1[i]];
    // 生成子密钥，保存在 subKeys[16] 中
    for (int round = 0; round<16; ++round)
    {
        // 前28位与后28位
        for (int i = 28; i<56; ++i)
            left[i - 28] = realKey[i];
        for (int i = 0; i<28; ++i)
            right[i] = realKey[i];
        // 左移
        left = leftShift(left, shiftBits[round]);
        right = leftShift(right, shiftBits[round]);
        // 压缩置换，由56位得到48位子密钥
        for (int i = 28; i<56; ++i)
            realKey[i] = left[i - 28];
        for (int i = 0; i<28; ++i)
            realKey[i] = right[i];
        for (int i = 0; i<48; ++i)
            compressKey[47 - i] = realKey[56 - PC_2[i]];
        subKey[round] = compressKey;
```

```
    }
}
```

将异或得到的 48 位数据分成 8 个 6 位的块,每一个块通过对应的一个 S 表产生一个 4 位的输出。其中，每个 S 表都是 4 行 16 列。具体的置换过程如下：把 6 位输入中的第 1 位和第 6 位取出来行成一个两位的二进制数 x ,作为 Si 表中的行数（0~3）；把 6 位输入的中间 4 位构成另外一个二进制数 y，作为 Si 表的列数（0~15）；查出 Si 表中 x 行 y 列所对应的整数，将该整数转换为一个 4 位的二进制数。

置换所用的 S 盒置换表

```
// S盒，每个S盒是4x16的置换表，对应6位 输入 4位输出
int S_BOX[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 }
        }
};
```

把通过 S 表置换得到的 8 个 4 位连在一起,形成一个 32 位的数据。然后将该 32 位数据通过表 P 进行置换（称为 P-置换），置换后得到一个仍然是 32 位的结果数据，这就是 f(R, K)函数的输出。

P 置换表

```
// P置换，32位 -> 32位
int P[] = {
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 };
```

```
// 密码函数f，接收32位数据和48位子密钥，产生一个32位的输出
bitset<32> f(bitset<32> R, bitset<48> k)
{
    bitset<48> expandR;
    // 扩展置换，32 -> 48
    for (int i = 0; i<48; ++i)
        expandR[47 - i] = R[32 - E[i]];
    // 异或运算
    expandR = expandR ^ k;
    // 查找S盒置换表
    bitset<32> output;
    int x = 0;
```

```
    for (int i = 0; i<48; i = i + 6)
    {
        int row = expandR[47 - i] * 2 + expandR[47 - i - 5];
        int col = expandR[47 - i - 1] * 8 + expandR[47 - i - 2] * 4 + expandR[47 - i - 3] *
2 + expandR[47 - i - 4];
        int num = S_BOX[i / 6][row][col];
        bitset<4> binary(num);
        output[31 - x] = binary[3];
        output[31 - x - 1] = binary[2];
        output[31 - x - 2] = binary[1];
        output[31 - x - 3] = binary[0];
        x += 4;
    }
    // P-置换，32 -> 32
    bitset<32> tmp = output;
    for (int i = 0; i<32; ++i)
        output[31 - i] = tmp[32 - P[i]];
    return output;
}
```

2.5 尾置换

合并 L16 和 R16 得到一个 64 位的数据，再经过尾置换后得到的就是 64 位的密文。注意：要将 L16 和 R16 合并成 R16L16（即左右互换）。

尾置换表 IP-1 如下：

```
// 尾置换表
int IP_1[] = {
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 };
```

```
//DES加密
bitset<64> encrypt(bitset<64>& plain)
{
    bitset<64> cipher;
    bitset<64> currentBits;
    bitset<32> left;
    bitset<32> right;
    bitset<32> newLeft;
    // 初始置换IP
    for (int i = 0; i<64; ++i)
        currentBits[63 - i] = plain[64 - IP[i]];
```

```
    // 获取 Li 和 Ri
    for (int i = 32; i<64; ++i)
        left[i - 32] = currentBits[i];
    for (int i = 0; i<32; ++i)
        right[i] = currentBits[i];
    // 共16轮迭代
    for (int round = 0; round<16; ++round)
    {
        newLeft = right;
        right = left ^ f(right, subKey[round]);
        left = newLeft;
    }
    // 合并L16和R16，注意合并为 R16L16
    for (int i = 0; i<32; ++i)
        cipher[i] = left[i];
    for (int i = 32; i<64; ++i)
        cipher[i] = right[i - 32];
    // 结尾置换IP-1
    currentBits = cipher;
    for (int i = 0; i<64; ++i)
        cipher[63 - i] = currentBits[64 - IP_1[i]];
    // 返回密文
    return cipher;
}
```

2.6 解密

DES 解密算法与加密算法完全相同，只需要将密文作为输入，秘钥使用加密时使用的密钥，注意使用子密钥时将使用顺序反过来。

```
//DES解密
bitset<64> decrypt(bitset<64>& cipher)
{
    bitset<64> plain;
    bitset<64> currentBits;
    bitset<32> left;
    bitset<32> right;
    bitset<32> newLeft;
    // 初始置换IP
    for (int i = 0; i<64; ++i)
        currentBits[63 - i] = cipher[64 - IP[i]];
    // 获取 Li 和 Ri
    for (int i = 32; i<64; ++i)
        left[i - 32] = currentBits[i];
    for (int i = 0; i<32; ++i)
        right[i] = currentBits[i];
    // 16轮迭代（子密钥逆序）
    for (int round = 0; round<16; ++round)
    {
```

```
            newLeft = right;
            right = left ^ f(right, subKey[15 - round]);
            left = newLeft;
        }
        // 合并L16和R16，注意合并为 R16L16
        for (int i = 0; i<32; ++i)
            plain[i] = left[i];
        for (int i = 32; i<64; ++i)
            plain[i] = right[i - 32];
        // 结尾置换IP-1
        currentBits = plain;
        for (int i = 0; i<64; ++i)
            plain[63 - i] = currentBits[64 - IP_1[i]];
        // 返回明文
        return plain;
    }
```

2.7 输出

将加密后的密文和解密后的明文分别存到文件ciphertext.txt和plain.txt中并打印到屏幕

```
int main() {
    string plaintext = "plain.txt";
    string crypt = "ciphertext.txt";
    string s = "12345678";
    cout << "请输入需要加密的明文plain: ";
    cin >> s;
    string k = "12345678";
    cout << "请输入8位秘钥K(数字/英文字母/符号|中文字符): ";
    cin >> k;
    int lens = s.length();
    int postion = 0;
    fstream file1;
    file1.open(crypt, ios::binary | ios::out);
    //解析明文，转换位64位二进制
    while (postion < lens) {
        char ende[8] = { ' ',' ',' ',' ',' ',' ',' ',' ' };
        int enlength = 0;
        if ((lens - postion) / 8 > 0) {
            enlength = 8;
        }
        else {
            enlength = lens - postion;
        }
        memcpy(ende, s.c_str() + postion, enlength);
        //cout << ch;
        postion = postion + 8;
```

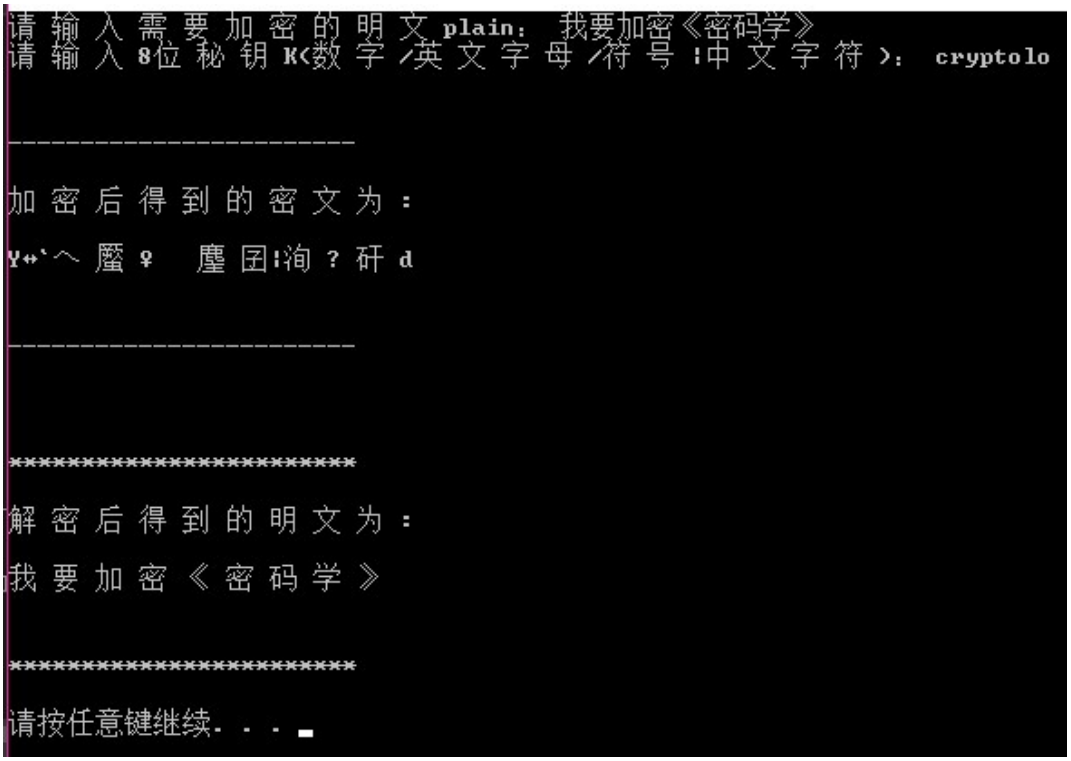
```
            bitset<64> plain = charToBitset(ende);
            key = charToBitset(k.c_str());
            // 生成16个子密钥
            generateKeys();
            // 密文写入crypt
            bitset<64> cipher = encrypt(plain);
            file1.write((char*)&cipher, sizeof(cipher));
            //cout << (char*)&cipher;
        }
        file1.close();

        // 读文件 crypt
        bitset<64> temp;
        file1.open(crypt, ios::binary | ios::in);
        fstream file2;
        file2.open(plaintext, ios::binary | ios::out);
        while (file1.read((char*)&temp, sizeof(temp))) {
            // 解密，并写入文件plaintext
            bitset<64> temp_plain = decrypt(temp);
            file2.write((char*)&temp_plain, sizeof(temp_plain));
        }
        file1.close();
        file2.close();
        cout << endl << endl << "------------------------" << endl << endl;
        cout << "加密后得到的密文为:" << endl<<endl;
        ifstream infile1;
        infile1.open(crypt);
        string content;
        while (getline(infile1, content)) {
            cout << content << endl;
        }
        cout << endl << endl << "------------------------" << endl << endl;
        infile1.close();
        cout << endl << endl << "************************" << endl << endl;
        cout << "解密后得到的明文为:" << endl<<endl;
        ifstream infile2;
        infile2.open(plaintext);
        while (getline(infile2, content)) {
            cout << content << endl;
        }
        cout << endl << endl << "************************" << endl << endl;
        infile2.close();
        return 0;
}
```

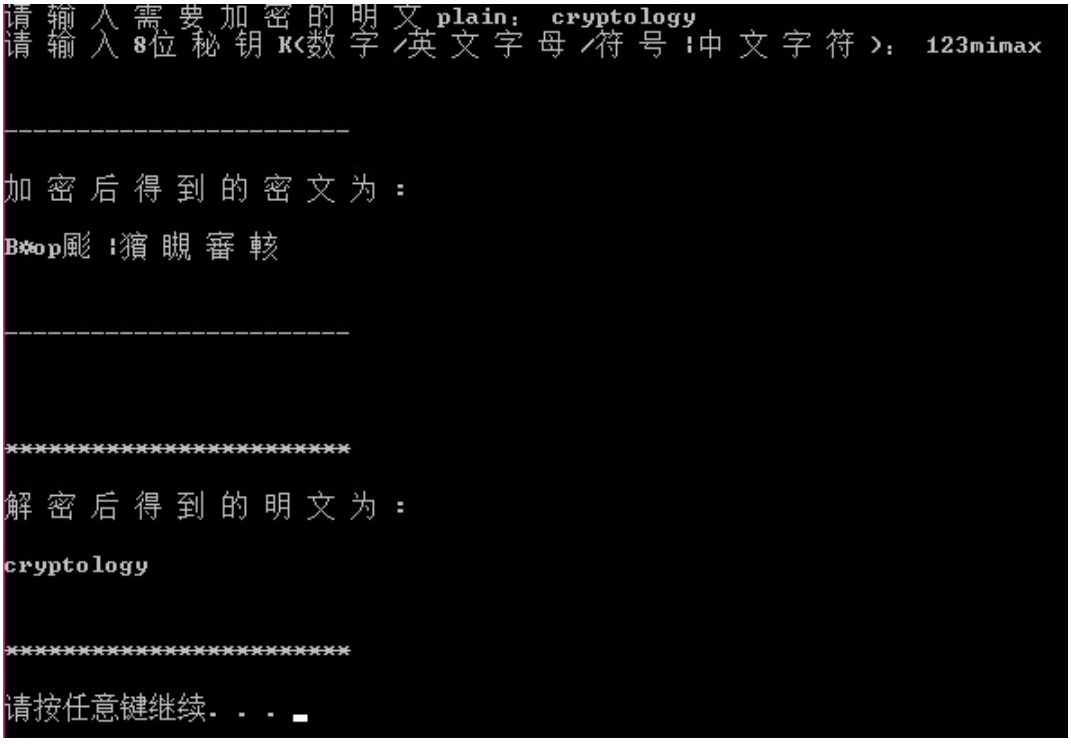
3、运行结果

运行结果如下

示例 1


请输入需要加密的明文plain: 我要加密《密码学》
请输入8位秘钥K(数字/英文字母/符号!中文字符): cryptolo
加密后得到的密文为:
解密后得到的明文为:
我要加密《密码学》
请按任意键继续. . .

示例 2


请输入需要加密的明文plain: cryptology
请输入8位秘钥K(数字/英文字母/符号!中文字符): 123mimax
加密后得到的密文为:
解密后得到的明文为:
cryptology
请按任意键继续. . .

示例 3

三 实验总结

DES 的加密结果可以看做是明文 m 和密钥 k 之间的一种复杂函数，即使是对明文和密钥的微小改变，产生的密文序列都将会发生很大的变化。

DES 运用了置换、替代、代数等多种密码技术，算法结构紧凑，条理清楚，而且加密和解密算法类似，便于工程实现。

四、附完整代码

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

bitset<64> key;                 // 64位密钥
bitset<48> subKey[16];          // 存放16轮子密钥

// 初始置换表
int IP[] = {
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 };

// 尾置换表
int IP_1[] = {
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 };

/*------------------生成密钥表-----------------*/

// 密钥置换表，将64位密钥变成56位
int PC_1[] = {
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 };

// 压缩置换，将56位密钥压缩成48位子密钥
int PC_2[] = {
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 };

// 每轮左移的位数
int shiftBits[] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

/*------------------函数 f 所用表-----------------*/
// 扩展置换表，将 32位 扩展至 48位
int E[] = {
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 };

// S盒，每个S盒是4x16的置换表，对应6位 输入 4位输出
int S_BOX[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 }
    }
};

// P置换，32位 -> 32位
int P[] = {
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 };

//DES 算法实现


// 密码函数f，接收32位数据和48位子密钥，产生一个32位的输出

bitset<32> f(bitset<32> R, bitset<48> k)
{
    bitset<48> expandR;
    // 扩展置换，32 -> 48
    for (int i = 0; i<48; ++i)
        expandR[47 - i] = R[32 - E[i]];
    // 异或运算
    expandR = expandR ^ k;
    // 查找S盒置换表
    bitset<32> output;
    int x = 0;
    for (int i = 0; i<48; i = i + 6)
    {
        int row = expandR[47 - i] * 2 + expandR[47 - i - 5];
        int col = expandR[47 - i - 1] * 8 + expandR[47 - i - 2] * 4 + expandR[47 - i - 3] *
2 + expandR[47 - i - 4];
```

```
            int num = S_BOX[i / 6][row][col];
            bitset<4> binary(num);
            output[31 - x] = binary[3];
            output[31 - x - 1] = binary[2];
            output[31 - x - 2] = binary[1];
            output[31 - x - 3] = binary[0];
            x += 4;
        }
        // P-置换，32 -> 32
        bitset<32> tmp = output;
        for (int i = 0; i<32; ++i)
            output[31 - i] = tmp[32 - P[i]];
        return output;
    }


    //对56位密钥的前后部分进行左移
    bitset<28> leftShift(bitset<28> k, int shift)
    {
        bitset<28> tmp = k;
        for (int i = 27; i >= 0; --i)
        {
            if (i - shift<0)
                k[i] = tmp[i - shift + 28];
            else
                k[i] = tmp[i - shift];
        }
        return k;
    }


    //生成16个48位的子密钥
    void generateKeys()
    {
        bitset<56> realKey;
        bitset<28> left;
        bitset<28> right;
        bitset<48> compressKey;
        // 去掉奇偶标记位，将64位密钥变成56位
        for (int i = 0; i<56; ++i)
            realKey[55 - i] = key[64 - PC_1[i]];
        // 生成子密钥，保存在 subKeys[16] 中
        for (int round = 0; round<16; ++round)
        {
            // 前28位与后28位
            for (int i = 28; i<56; ++i)
                left[i - 28] = realKey[i];
            for (int i = 0; i<28; ++i)
                right[i] = realKey[i];
            // 左移
            left = leftShift(left, shiftBits[round]);
            right = leftShift(right, shiftBits[round]);
```

```
        // 压缩置换，由56位得到48位子密钥
        for (int i = 28; i<56; ++i)
            realKey[i] = left[i - 28];
        for (int i = 0; i<28; ++i)
            realKey[i] = right[i];
        for (int i = 0; i<48; ++i)
            compressKey[47 - i] = realKey[56 - PC_2[i]];
        subKey[round] = compressKey;
    }
}

//将char字符数组转为二进制
bitset<64> charToBitset(const char s[8])
{
    bitset<64> bits;
    for (int i = 0; i<8; ++i)
        for (int j = 0; j<8; ++j)
            bits[i * 8 + j] = ((s[i] >> j) & 1);
    return bits;
}


//DES加密
bitset<64> encrypt(bitset<64>& plain)
{
    bitset<64> cipher;
    bitset<64> currentBits;
    bitset<32> left;
    bitset<32> right;
    bitset<32> newLeft;
    // 初始置换IP
    for (int i = 0; i<64; ++i)
        currentBits[63 - i] = plain[64 - IP[i]];
    // 获取 Li 和 Ri
    for (int i = 32; i<64; ++i)
        left[i - 32] = currentBits[i];
    for (int i = 0; i<32; ++i)
        right[i] = currentBits[i];
    // 共16轮迭代
    for (int round = 0; round<16; ++round)
    {
        newLeft = right;
        right = left ^ f(right, subKey[round]);
        left = newLeft;
    }
    // 合并L16和R16，注意合并为 R16L16
    for (int i = 0; i<32; ++i)
        cipher[i] = left[i];
    for (int i = 32; i<64; ++i)
        cipher[i] = right[i - 32];
    // 结尾置换IP-1
    currentBits = cipher;
```

```
    for (int i = 0; i<64; ++i)
        cipher[63 - i] = currentBits[64 - IP_1[i]];
    // 返回密文
    return cipher;
}

//DES解密
bitset<64> decrypt(bitset<64>& cipher)
{
    bitset<64> plain;
    bitset<64> currentBits;
    bitset<32> left;
    bitset<32> right;
    bitset<32> newLeft;
    // 初始置换IP
    for (int i = 0; i<64; ++i)
        currentBits[63 - i] = cipher[64 - IP[i]];
    // 获取 Li 和 Ri
    for (int i = 32; i<64; ++i)
        left[i - 32] = currentBits[i];
    for (int i = 0; i<32; ++i)
        right[i] = currentBits[i];
    // 16轮迭代（子密钥逆序）
    for (int round = 0; round<16; ++round)
    {
        newLeft = right;
        right = left ^ f(right, subKey[15 - round]);
        left = newLeft;
    }
    // 合并L16和R16，注意合并为 R16L16
    for (int i = 0; i<32; ++i)
        plain[i] = left[i];
    for (int i = 32; i<64; ++i)
        plain[i] = right[i - 32];
    // 结尾置换IP-1
    currentBits = plain;
    for (int i = 0; i<64; ++i)
        plain[63 - i] = currentBits[64 - IP_1[i]];
    // 返回明文
    return plain;
}


int main() {
    string plaintext = "plain.txt";
    string crypt = "ciphertext.txt";
    string s = "12345678";
    cout << "请输入需要加密的明文plain：";
    cin >> s;
    string k = "12345678";
    cout << "请输入8位秘钥K(数字/英文字母/符号|中文字符)：";
    cin >> k;
```

```
    int lens = s.length();
    int postion = 0;
    fstream file1;
    file1.open(crypt, ios::binary | ios::out);
    //解析明文，转换位64位二进制
    while (postion < lens) {
        char ende[8] = { ' ',' ',' ',' ',' ',' ',' ',' ' };
        int enlength = 0;
        if ((lens - postion) / 8 > 0) {
            enlength = 8;
        }
        else {
            enlength = lens - postion;
        }
        memcpy(ende, s.c_str() + postion, enlength);
        //cout << ch;
        postion = postion + 8;
        bitset<64> plain = charToBitset(ende);
        key = charToBitset(k.c_str());
        // 生成16个子密钥
        generateKeys();
        // 密文写入crypt
        bitset<64> cipher = encrypt(plain);
        file1.write((char*)&cipher, sizeof(cipher));
        //cout << (char*)&cipher;
    }
    file1.close();

    // 读文件 crypt
    bitset<64> temp;
    file1.open(crypt, ios::binary | ios::in);
    fstream file2;
    file2.open(plaintext, ios::binary | ios::out);
    while (file1.read((char*)&temp, sizeof(temp))) {
        // 解密，并写入文件plaintext
        bitset<64> temp_plain = decrypt(temp);
        file2.write((char*)&temp_plain, sizeof(temp_plain));
    }
    file1.close();
    file2.close();
    cout << endl << endl << "-------------------------" << endl << endl;
    cout << "加密后得到的密文为:" << endl<<endl;
    ifstream infile1;
    infile1.open(crypt);
    string content;
    while (getline(infile1, content)) {
        cout << content << endl;
    }
    cout << endl << endl << "-------------------------" << endl << endl;
    infile1.close();
    cout << endl << endl << "************************" << endl << endl;
    cout << "解密后得到的明文为:" << endl<<endl;
```

```
    ifstream infile2;
    infile2.open(plaintext);
    while (getline(infile2, content)) {
        cout << content << endl;
    }
    cout << endl << endl << "**********" << endl << endl;
    infile2.close();
    return 0;
}
```