杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

《密码学实验》

实验报告

实验 3：DES 分组密码算法编程实验

姓　　名：易涛

学　　号：16031330

专　　业：信息安全

实验时间：2018.04.16

指导老师：吕秋云

一．实验目的

实验环境：Windows10 Visual Studio 2017

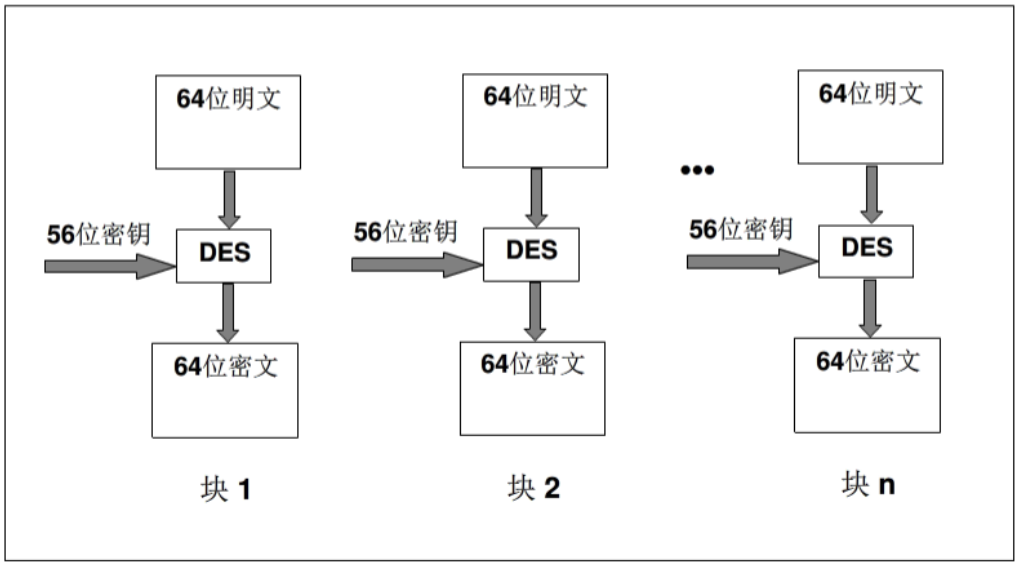
实验目的：

掌握 DES 密码加解密原理，并利用 Visual C++编程实现

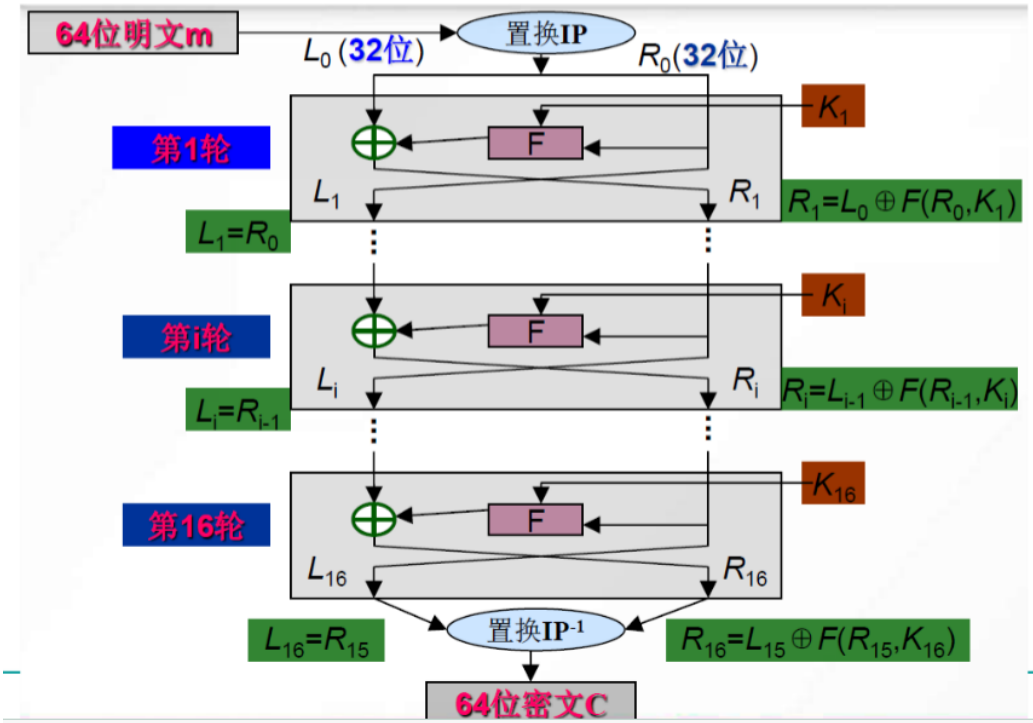
加密原理：

DES 是一种分组加密算法，每组的明文长度是 64 bit，密钥长度为 56bit ，加密后的密文长度也是 64 bit。实际中的明文未必恰好是 64 bit，所以要经过分组和填充把它们对齐为若干个 64bit 的组，然后进行加密处理。解密过程则相反，它首先按照分组进行解密，然后去除填充信息并进行连接，DES 的工作原理下图所示。

■ DES的工作原理



DES 的主体运算由初始置换、Feistel 网络组成。整体逻辑如下图所示。



实验要求：

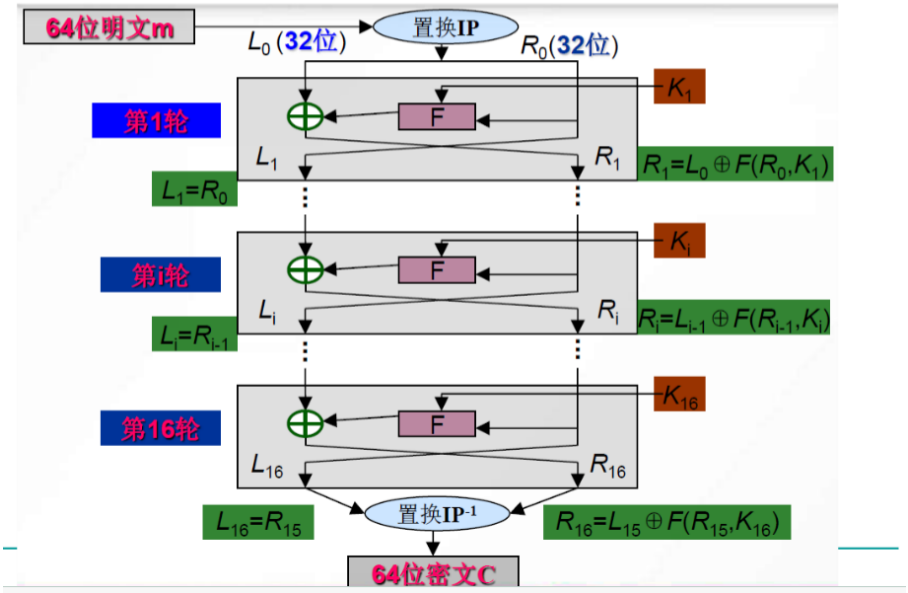
1> 采用 MFC 编程实现简单页面编程

2> 模块化编程

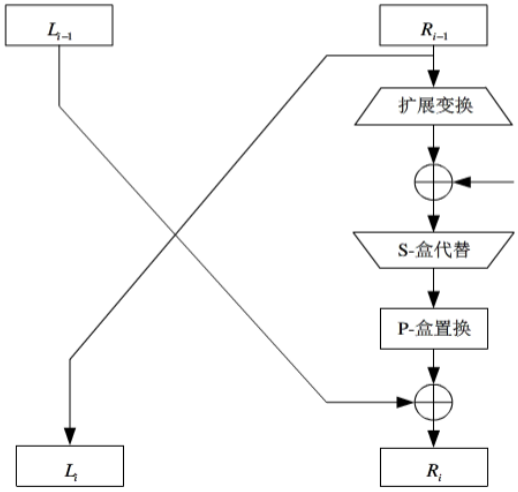
3> 通过 DES 实现对不同文件格式的文件加解密

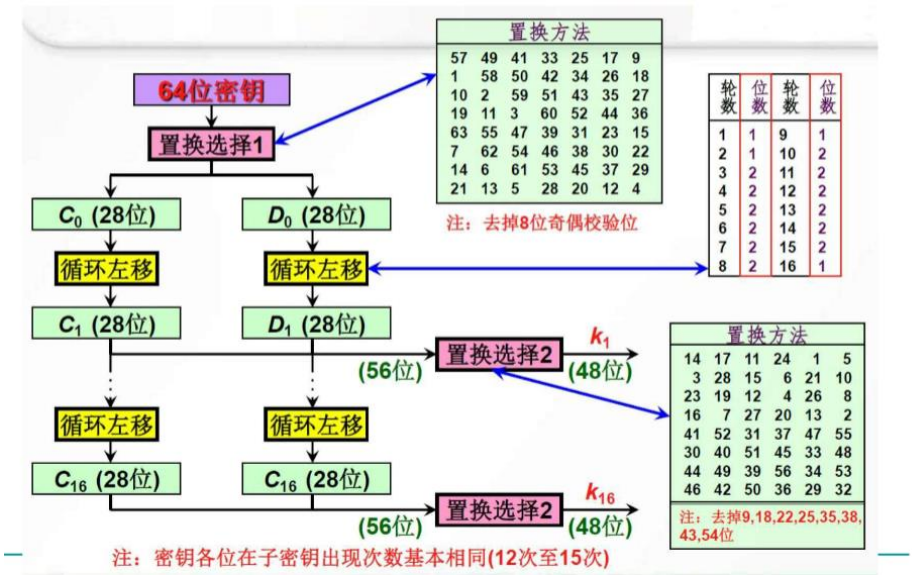
二．实验内容及实现过程步骤

1> DES 的加密步骤如图，因为采用模块化编程，故将 DES 中主要步骤操作各编写成子函数，供上一层函数调用。



- 圈函数结构图：





2> 由上图分析可得，为实现模块化编程，以及使程序编写更严谨有逻辑，需定义下列子函数，因为存在底层调用，故先声明所有所需函数（并不编写函数代码），再由顶层向底层编写。

```
void Transform(bool* Out, bool* In, const char* Table, int len);//表置换
void Des_Run(char Out[8], char In[8], bool flag);//加解密函数，靠flag选择功能
void Des_SetSubkey(char key[8]);//生成子密钥
void F_func(bool In[32], bool Ki[48]);//F轮函数
void S_func(bool Out[32], bool In[48]);//S盒置换
void Xor(bool* InA, bool* InB, int len);//异或
void Rotatel(bool* In, int len, int loop);//循环左移
void ByteToBit(bool* Out, const char* In, int bits);//字符转比特
void HalfByteToBit(bool* Out, const char* In, int bits);//半字符转比特
void BitToByte(char* Out, const bool* In, int bits);//比特转字符
void Des_File(char *msgFile, char *cipherFile); //文件加密，从文本框获取路径
void Des_File_D(char *msgFile, char *cipherFile);//文件解密，从文本框读入文件路径
```

3> DES 顶层函数，DES_Run，DES_Run 通过对所有底层模块的调用实现 DES 加密和解密，代码如下图所示，参数由左至右分别表示：经 DES 加密或解密后的输出，需要 DES 加密或解密的输入，模式选择（flag 为 1 则为加密，为 0 为解密）。

此函数流程为

1.先将输入的 8 个字符转化成 64 bit 储存在 M 数组中，通过 ByteToBit 函数实现，同时将 M 分为左右两组，分别是 Li，Ri

2.通过 flag 选择加密还是解密功能，加解密代码类似，只是将所需的轮密钥逆序，故只叙述加密代码

3.加密代码，通过 Transform 函数将 M 进行初始置换 IP，置换完成之后输出重新写入 M 中，故 1 中的 Li、Ri 还代表 M 的左右分组。之后进行 16 轮运算，即：$L_i = R_{i-1}$，$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$，16 轮运算后，经 IP 逆置换，然后将 64 bit 转成字符存到 Out 字符数组里输出，加密完成。

```
void Des_Run(char Out[8], char In[8], bool flag)
{
    static bool M[64];//存储64位明文，由明文字符转换得来
    static bool Temp[32];//中间变量，用于对Li赋值
    static bool* Li = &M[0];//分为左右两组
    static bool* Ri = &M[32];
    ByteToBit(M, In, 64);
    if (flag)
    {
        Transform(M, M, IP_Table, 64);//初始置换IP
        for (int i = 0; i < 16; i++)
        {
            memcpy(Temp, Ri, 32);//把R(i-1)保存起来
            F_func(Ri, SubKey[i]);
            Xor(Ri, Li, 32);
            memcpy(Li, Temp, 32);//Li=R(i-1)
        }
        memcpy(Temp, Li, 32);
        memcpy(Li, Ri, 32);
        memcpy(Ri, Temp, 32);//最后交换Li,Ri
        Transform(M, M, IP_N_Table, 64);

    }
    else {
        Transform(M, M, IP_Table, 64);
        for (int i = 15; i >= 0; i--)//没有等于0
        {
            memcpy(Temp, Ri, 32);//把R(i-1)保存起来
            F_func(Ri, SubKey[i]);
            Xor(Ri, Li, 32);
            memcpy(Li, Temp, 32);//Li=R(i-1)
        }
        memcpy(Temp, Li, 32);
        memcpy(Li, Ri, 32);
        memcpy(Ri, Temp, 32);//最后交换Li,Ri
        Transform(M, M, IP_N_Table, 64);
    }
    BitToByte(Out, M, 64);
}
```

4> DES 加密过程已在 3 中说明，接下来说明 DES 加密过程中调用的底层函数，从 DES_Run 函数由上到下，首先是 ByteToBit 函数，输入字符数组 In，需转化的 bit 数，输出 bool 比特数组 Out。

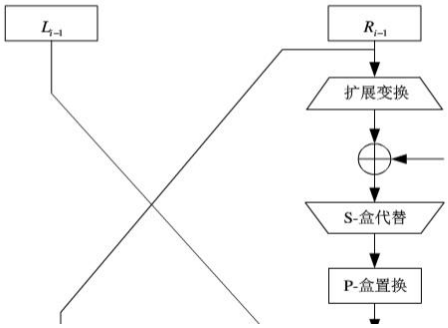
```
}
void ByteToBit(bool* Out, const char* In, int bits)
{
    //把各种盒或表里的十进制转成比特
    for (int i = 0; i < bits; i++)
    {
        Out[i] = (In[i / 8] >> (i % 8)) & 1;
    }
}
```

Transform 函数，此函数用于 DES 加密过程中的置换，包括 IP 置换，IP 逆置换，轮函数中的扩展变换 E、P 盒置换，密钥扩展中的 PC-1 置换、PC-2 置换。事先将各种置换矩阵定义成数组，通过参数 const char* Table 传入。因为加密过程中，最大的置换是输出 64 bit，故定义一个中间 bool 数组，长度为 100>64，置换代码很简单，调用时传入置换后输出的比特串长度 len，从 0 循环到 len-1，对每一轮 i，将置换矩阵中对应数值作为 In 的下标即 In[Table[i]-1]，Temp[i] = In[Table[i]-1]进行赋值完成置换，循环之后将 Temp 中 len 长度的数据赋值给 Out 输出。

```
void Transform(bool* Out, bool* In, const char* Table, int len)//置换
{
    bool Temp[100];
    for (int i = 0; i < len; i++)
    {
        //Out[i] = In[Table[i] - 1];//Table[i]-1，从0开始计数
        Temp[i] = In[Table[i] - 1];
    }
    memcpy(Out, Temp, len);
}
```

F_func 函数，如 F 函数结构图所示，F_func 参数输入由R_{i-1}和每一轮密钥 Ki 组成，依次进行扩展变换 E，得到 MR 与轮密钥 Ki 异或，在将得到的 MR 与 In 传入 S 函数，S 函数进行 S 盒替代，替代完成的结果放在 In 数组中，再将 In 进行 P 置换完成 F 函数。

- 圈函数结构图：




```
void F_func(bool In[32], bool Ki[48])
{
    static bool MR[48];
    Transform(MR, In, E_Table, 48);
    Xor(MR, Ki, 48);
    S_func(In, MR);
    Transform(In, In, P_Table,32);

}
```

Xor 函数，异或，输入需异或的 InA,InB 比特串还有需异或的比特数，然后按位异或，将异或后的结果保存在 InA 中

```
void Xor(bool* InA, bool* InB, int len)
{
    for (int i = 0; i < len; i++)
        InA[i] = (InA[i] ^ InB[i]);
}
```

S_func 函数，S 函数主要做 S 盒代替功能，输入 48 比特，输出经 S 盒代替的 32 比特，将输入的 48 比特分成 8 组，每组 6 比特，每 6 比特的最高最低位组成行号，中间 4 位组成列号，再将 S 盒中对应组行列的数转化成 4 比特，逐次存入 Out 数组中。

```
void S_func(bool Out[32], bool In[48])//错误写成了bool* Out[32]
{
    for (char i = 0, j, k; i < 8; i++)//
    {
        j = (In[0+6*i] << 1) + In[5+6*i];//行等于首尾拼起来组成的十进制数
        k = (In[1+6*i] << 3) + (In[2+6*i] << 2) + (In[3+6*i] << 1) + In[4+6*i];
        HalfByteToBit(Out+4*i, &S_Box[i][j][k], 4);
    }
}
```

BitToByte 函数，参数有 char 型 Out 数组，bool 型 In 比特串，比特数，功能将比特串转化为字符串，8bit 转化一个字符。

```
void BitToByte(char* Out, const bool* In, int bits)
{
    memset(Out, 0, bits >> 3);//初始化8个字节的内存
    for (int i = 0; i < bits; i++)
    {
        Out[i >> 3] |= In[i] << (i & 7);
    }
}
```

5> DES 加密所用到函数都已实现，接下来是编写生成 16 轮子密钥的函数。

DES_SetSubKey，该函数输入为初始密钥 char key[8]，首先将 key 转化成 64 比特存在 K 数组中，再进行 PC_1 置换，置换得到 56 比特仍然放在 K 数组中，将 56 比特均分为 KL，KR 左右两部分，然后开始 16 轮循环生成 16 轮子密钥，循环过程中将 KL，KR 进行移位，每一轮移位位数保存在 LS_Table 里，移位之后将 K 进行 PC_2 置换，生成对应 48bit 子密钥。

```
void Des_SetSubkey(char key[8])
{
    static bool K[64];
    static bool* KL = &K[0];
    static bool* KR = &K[28];
    ByteToBit(K, key, 64);
    Transform(K, K, PC_1_Table, 56);
    for (int i = 0; i < 16; i++)
    {
        Rotatel(KL, 28, LS_Table[i]);
        Rotatel(KR, 28, LS_Table[i]);
        Transform(SubKey[i], K, PC_2_Table,48);//忘记PC_2置换了

    }
}
```

Rotatel 函数，将输入比特串 In 进行循环移位，总长为 len，移位数为 loop

```
void Rotatel(bool* In, int len, int loop)
{
    static bool Temp[28];
    memcpy(Temp, In, loop);//循环位loop位保存在temp中
    memcpy(In, In + loop, len - loop);//左移
    memcpy(In + len - loop, Temp, loop);//补上
}
```

6> 加密过程和生成密钥代码均已编写完成，接下来编写函数对文件进行加解密

DES_Flie 函数，参数为两个字符串，分别表示待处理待生成文件的路径，通过 ifstream 和 ofstream 读取待处理文件，写入待生成文件。将待处理文件按每 8 字节读入 msgBlock 中，将 msgBlock 进行 des 加密得到 cipherBlock，将 cipherBlock 写入待生成文件。

```
void Des_File(char *msgFile, char *cipherFile)
{   // ...
    char msgBlock[8], cipherBlock[8];
    //int count;
    ifstream msg(msgFile, ios::binary);//读文件
    ofstream cipher(cipherFile, ios::binary);//写文件

    while (/*!msg.eof()*/msg.peek() != EOF)//出错解决   //c++ eof会多读一次
    {
        /* ... */
        memset(msgBlock,0,8);//先默认为0，省去填充
        msg.read(msgBlock, 8);
        Des_Run(cipherBlock, msgBlock, 1);
        cipher.write(cipherBlock, 8);
    }
    /* ... */
    msg.close();
    cipher.close();
}
```

DES_File_D 函数，与 DES_File 函数类似，不多叙述。

```
void Des_File_D(char *msgFile, char *cipherFile)
{   // ...

    char msgBlock[8], cipherBlock[8];

    // ...

    ofstream msg(msgFile, ios::binary);
    ifstream cipher(cipherFile, ios::binary);
    //int count;
    while (cipher.peek() != EOF)
    {
        /* ... */
        //出错，数组越界cipherBlock[8] = { 0 };
        memset(cipherBlock, 0, 8);
        cipher.read(cipherBlock, 8);
        Des_Run(msgBlock, cipherBlock, 0);
        msg.write((char*)&msgBlock, 8);

    }
    msg.close();
    cipher.close();
}
```

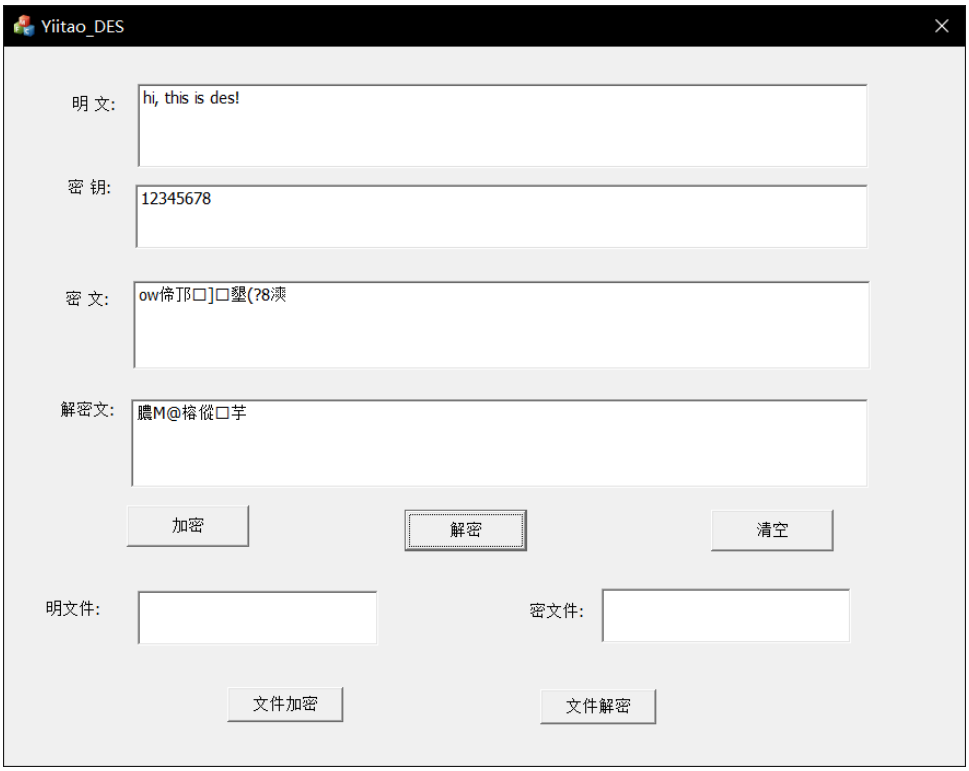
三．实验小结

1> 实验中遇到的错误

1. 程序编写完成之后，输入明文和密钥进行加密，加密得到的密文解密还原不了，解密无反应。
   解决方法：
   在读入明文和密钥之后设断点，开启调试，先从加密过程逐语句逐函数运行，运行到 F 函数的扩展变化 E 的时候发现数据异常，于是检查 E 变化矩阵，发现 E 变化矩阵少输入一行，造成错误。于是将 E 置换矩阵补充完全。此外重新检查一遍各置换矩阵的数据，发现 S 盒中一个数据输入错误。

```
45      };
46      //扩展表E
47      static const char E_Table[48] = {
48          //少打一行
49          32, 1, 2, 3, 4, 5,
50          4, 5, 6, 7, 8, 9,
51          8, 9, 10, 11, 12, 13,
52          12, 13, 14, 15, 16, 17,
53          16, 17, 18, 19, 20, 21,
54          /**/20, 21, 22, 23, 24, 25,
55          24, 25, 26, 27, 28, 29,
56          28, 29, 30, 31, 32, 1
57      };
58      //S盒
```

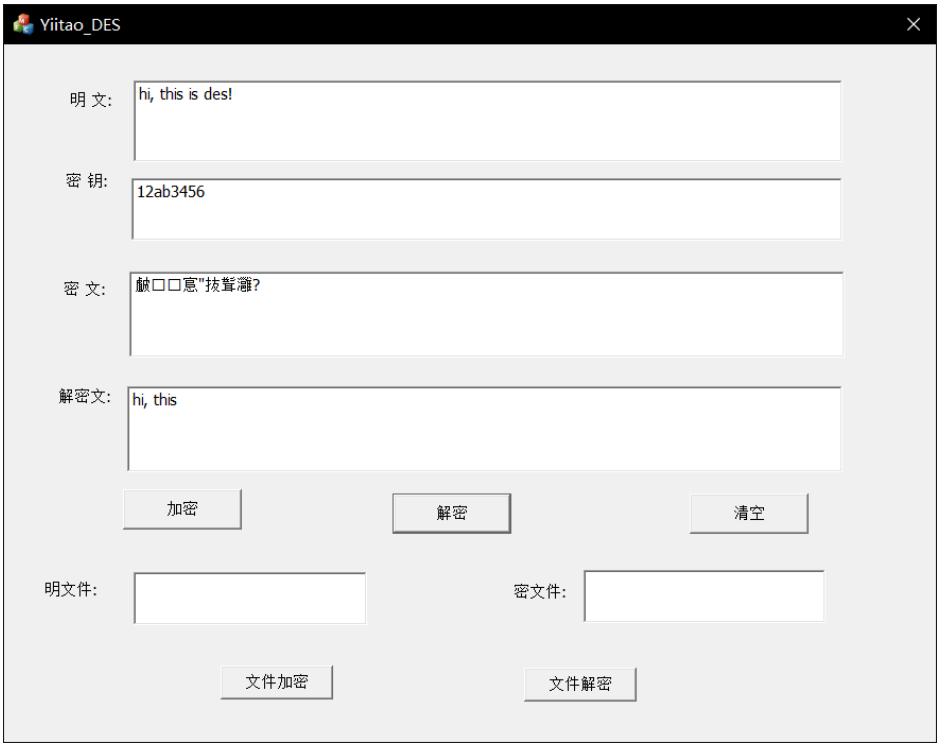
2. 解决 1 中解密无反应的错误之后，输入明文密钥加密得到密文，将密文解密，发现解密的数据错误，与明文不符。。



解决方法：加密过程确认无其他错误，错误可以定位于密钥设置和解密函数之间，先检查解密函数，设断点于解密代码处，开启调试，程序在解密代码开始处中断，逐步调试，发现解密轮数少了一轮，循环语句出错，将语句修改解决。

```
185     else {
186         Transform(M, M, IP_Table, 64);
187         for (int i = 15; i > 0; i--)//没有等于0
188         {
189             memcpy(Temp, Ri, 32);//把R(i-1)保存起来
190             F_func(Ri, SubKey[i]);
191             Xor(Ri, Li, 32);
192             memcpy(Li, Temp, 32);//Li=R(i-1)  已用时间 <= 1ms
193         }
194         memcpy(Temp, Li, 32);
```

3. 解决 2 中问题之后，再运行程序，发现解密得到的解密文总是比输入的明文少两位。



解决方法：加密代码和解密代码确认没有问题，将断点设置在生成子密钥代码处，开启调试，发现代码编写时，漏了一个步骤 PC_2 置换，于是将语句添上。

```
257
258 void Des_SetSubkey(char key[8])
259 {
260     static bool K[64];
261     static bool* KL = &K[0];
262     static bool* KR = &K[28];
263     ByteToBit(K, key, 64);
264     Transform(K, K, PC_1_Table, 56);
265     for (int i = 0; i < 16; i++)
266     {
267         Rotatel(KL, 28, LS_Table[i]);
268         Rotatel(KR, 28, LS_Table[i]);  已用时间 <= 1ms
269         //Transform(SubKey[i], K, PC_2_Table,48);//忘记PC_2置换了
270
271     }
272 }
```

4. 解决 1、2、3 错误之后，对输入字符串进行 des 加解密可以实现，开始对文件操作进行编写，一开始使用 c 语言中的 FILE 文件指针，以及 read、write 函数，结果编译时语法报错，说 read、write 为 unsafe 函数，于是改用 c++文件流，ifstream 和 ofstream 对文件进行读写。

```
void Des_File(char *msgFile, char *cipherFile)
{   //文件加密，从文本框获取路径
    //FILE * msg, *cipher;
    char msgBlock[8], cipherBlock[8];
    //int count;
    ifstream msg(msgFile, ios::binary);//读文件
    ofstream cipher(cipherFile, ios::binary);//写文件

    while (/*!msg.eof()*/msg.peek() != EOF)//出错解决   //c++ eof会多读一次
    {
        /*if ((count = fread(msgBlock, sizeof(char), 8, msg)) == 8)
        {
            Des_Run(cipherBlock, msgBlock, 1);
            fwrite(cipherBlock, sizeof(char), 8, cipher);
        }*/
        memset(msgBlock,0,8);//先默认为0，省去填充
        msg.read(msgBlock, 8);
        Des_Run(cipherBlock, msgBlock, 1);
        cipher.write(cipherBlock, 8);
    }
```

5. 解决 4 之后编译语法通过，但在进行文件加密解密时，发生错误，加密可以快速生成加密文件，但解密时，程序崩溃

解决方法：解密时出现问题，将问题定位至文件解密函数，设断点调试，发现在初始化密文数组块的时候出现错误，出现下图所圈语句导致数组越界，解决方法，用 memset 函数代替初始化。

```
326     ifstream cipher(cipherFile, ios::binary);
327     //int count;
328     while (cipher.peek() != EOF)
329     {
330         /*if ((count = fread(cipherBlock, sizeof(char), 8, ciphe
331         {
332             Des_Run(cipherBlock, msgBlock, 0);
333             fwrite(msgBlock, sizeof(char), 8, msg);
334         }*/
335         /*出错，数组越界*/cipherBlock[8] = { 0 };
336         //memset(cipherBlock, 0, 8);
337         cipher.read(cipherBlock, 8);
338         Des_Run(msgBlock, cipherBlock, 0);
339         msg.write((char*)&msgBlock, 8);
```

6. 解决 5 的错误之后，对文件进行加解密操作，发现加解密文件不一致，于是设断点对文件加解密代码进行调试，发现循环条件总是多循环一轮，于是上网查询 c++中的 eof 函数，发现 c++中 eof 函数设计上存在的问题。

fstream流的eof()判断有点不合常理

按常理逻辑来说，如果到了文件末尾的话，eof()应该返回真，但是，C++输入输出流如何知道是否到末尾了呢?

原来根据的是：如果fin>>不能再读入数据了，才发现到了文件结尾，这时才给流设定文件结尾的标志，此后调用eof()时，才返回真。

假设

find>>x; //此时文件刚好读完最后一个数据（将其保存在x中）

但是，这时fin.eof()仍为假，因为 fin流的标志eofbit是False，fin流此时认为文件还没有到末尾，只有当流再次读写时fin>>x，发现已无可读写数据，此时流才知道到达了结尾，这时才将标志eofbit修改为True，此时流才知道文件到了末尾。

也就是说，eof在读取完最后一个数据后，仍是False，当再次试图读一个数据时，由于发现没数据可读了，才知道到末尾了，此时才修改标志，eof变为True。

```
void Des_File(char *msgFile, char *cipherFile)
{   //文件加密，从文本框获取路径
    //FILE * msg, *cipher;
    char msgBlock[8], cipherBlock[8];
    //int count;
    ifstream msg(msgFile, ios::binary);//读文件
    ofstream cipher(cipherFile, ios::binary);//写文件

    while (!msg.eof()/*msg.peek() != EOF*/)//出错解决   //c++ eof会多读一次
    {
        /* ... */
        memset(msgBlock,0,8);//先默认为0，省去填充
        msg.read(msgBlock, 8);
        Des_Run(cipherBlock, msgBlock, 1);
        cipher.write(cipherBlock, 8);
    }
    /* ... */
    msg.close();
    cipher.close();
}
```

解决方法：不用 eof 函数判断文件是否读完，改用 peek 函数。

2> 实验总结

本次实验为 DES 加密实验，过程较为复杂，需学会模块化编程，将复杂问题简单化，分而治之；此外编程时间久，断断续续编程需多做注释，这样不至于出太多错误；另外当程序报错时，需分析可能出错的位置，并在可能出错的地方设置断点调试，一步步解决。