Assignment 1 实验报告

一. 算法原理概述

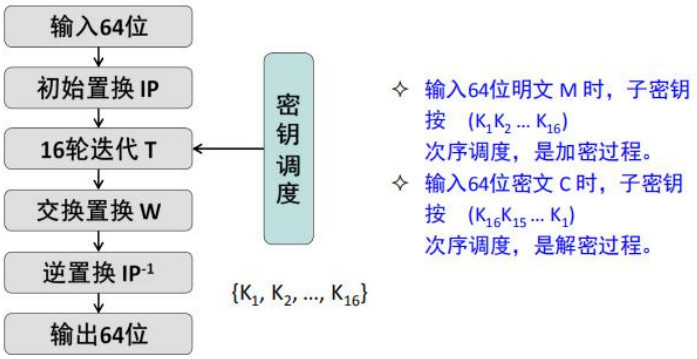
DES 算法是一种典型的对称加密算法:它以 64 位为分组长度，64 位一组的明文作为算法的输入，通过一系列复杂的操作，输出同样 64 位长度的密文.

DES 使用加密密钥定义变换过程，因此算法认为只有持有加密所用的密钥的用户才能解密密文。

DES 采用 64 位密钥，但由于每 8 位中的最后 1 位用于奇偶校验，实际有效密钥长度为 56 位。密钥可以是任意的 56 位的数，且可随时改变。其中极少量的数被认为是弱密钥，但能容易地避开它们。所有的保密性依赖于密钥。

DES 算法的基本过程是换位和置换。

二. 总体结构



上图为算法流程图

加密过程

$C = E_k(M) = IP^{-1} \cdot W \cdot T_{16} \cdot T_{15} \cdot \cdots \cdot T_1 \cdot IP(M)$.

(1). M 为算法输入的 64 位明文块;

(2). E_k 描述以 K 为密钥的加密函数，由连续的过程复合构成;

(3). IP 为 64 位初始置换;

(4). T_1, T_2 , …, T_{16} 是一系列的迭代变换;

(5). W 为 64 位置换，将输入的高 32 位和低 32 位交换后输出

(6). IP^{-1} 是 IP 的逆置换;

(7). C 为算法输出的 64 位密文块。

解密过程

$M = D_k(C) = IP^{-1} \cdot W \cdot T_1 \cdot T_2 \cdot \cdots \cdot T_{16} \cdot IP\ (C)$

1. 对于输入的 64 位明文首先进行 IP 置换，IP 置换目的是将输入的 64 位数据块按位重新组合，并把输出分为 L_0、R_0 两部分，每部分各长 32 位。

2. 进行 16 轮迭代 T，根据 LORO 按下述规则进行 16 次迭代，即
$L_i = R_{i-1}$　　$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$, $i = 1 .. 16$.
其中 f 是输出 32 位的 Feistel 轮函数
16 个长度为 48 位的子密钥 K_i（$i = 1 .. 16$）由密钥 K 生成；

3. 交换置换 W，将迭代结果的左右部分相互调转输出

4. 逆置换 IP^{-1}，对迭代 T 输出的二进制串 $R_{16}L_{16}$ 使用初始置换的逆置换 IP^{-1} 得到密文 C，即：$C = IP^{-1}(R_{16}L_{16})$

5. Feistel 轮函数 $f(R_{i-1}, K_i)$包括 5 个步骤:
(1) 将长度为 32 位的串 R_{i-1} 作 E-扩展，成为 48 位的串 $E(R_{i-1})$
(2) 将 $E(R_{i-1})$ 和长度为 48 位的子密钥 Ki 作 48 位二进制串按位异或运算，Ki 由密钥 K 生成；
(3) 将 (2) 得到的结果平均分成 8 个分组，每个分组长度 6 位。各个分组分别经过 8 个不同的 S-盒进行 6-4 转换，得到 8 个长度分别为 4 位的分组；
(4) 将 (3) 得到的分组结果顺序连接得到长度为 32 位的串
(5) 将 (4) 的结果经过 P-置换，得到的结果作为轮函数 $f(R_{i-1}, K_i)$ 的最终 32 位输出.

6. 关于子密钥的生成，用于 Feistel 轮函数。
(1) 对 K 的 56 个非校验位实行置换 P_{C-1}，得到 C_0D_0，其中 C_0 和 D_0 分别由 P_{C-1} 置换后的前 28 位和后 28 位组成。i = 1。
(2) 计算 $C_i = LS_i(C_{i-1})$ 和 $D_i = LS_i(D_{i-1})$ 当 i =1, 2, 9, 16 时，$LS_i(A)$ 表示将二进制串 A 循环左移一个位置；否则循环左移两个位置。
(3) 对 56 位的 C_iD_i 实行 PC-2 压缩置换，得到 48 位的 K_i 。i = i+1。
(4) 如果已经得到 K_{16}，密钥调度过程结束；否则转 (2)。

三. 模块分解

根据算法的整体结构，我将 DES 加密算法分成 Feistel 轮函数，生成子密钥，IP 置换，W 交换置换，IP 逆置换，输入输出，字符串转 bitset 这七个模块。

```
class DES
{
public:
    DES();
    ~DES();

    bitset<64> encrypt(bitset<64> &M);  // 加密函数，返回密文
    bitset<64> decrypt(bitset<64> &C);  // 解密函数，返回明文

    void show_encrypt(string m, string k);  // 将用户输入的字符串转换成二进制加密
    void show_decrypt(bitset<64> c, string k);  // 将用户输入的字符串转换成二进制解密

    string getM_str();
    string getK_str();
    bitset<64> getK();
    void setM_str(string temp);
    void setK_str(string temp);


private:
    string m_str;           // 明文字符串
    string k_str;           // 密钥字符串
    bitset<64> K;           // 64位密钥
    bitset<48> sub_K[16];   // 16个子密钥

    bitset<32> Feistel(bitset<32> R, bitset<48> supkey);  // Feistel 函数
    void generateKeys();    //生成16个48bit的子密钥,用于Feistel函数
    bitset<28> leftshift(bitset<28> k, int shift_num);  //左移函数,用于生成子密钥
    bitset<64> char_to_bitset(const char s[8]);   //将char字符转换为2进制的bitset
    string bitset_to_string(bitset<64> m);   //将2进制的bitset转换为char字符
};
```

以上为 DES 类的公有成员函数以及私有成员函数

1. 对于 Feistel 轮函数模块

先要对于传入的 32 位 bit 进行 E 盒扩展成 48 位，然后与子密钥 K 做按位异或运算。

```
// 将R作E扩展，成为48位的串E(R)
for (int i = 0; i < 48; ++i)
{
    expand[i] = R[E_Extension_Table[i] - 1];
}
// 将扩展后的E(R)和子密钥K作48位按位异或运算
expand ^= supkey;
```

然后分组，对每一个分组的 6 个 bit 通过 S 盒转换为 4 位，然后按顺序连接回 32 位 bit。

```
// 将上面结果平均分成8个分组，每个分组长度为6，进行S盒转换为4位
int output_index = 0;
for (int i = 0; i < 48; i = i + 6)
{
    // 假设 Si 的6位输入为 b1b2b3b4b5b6，
    // 则由 n = (b1b6)10 确定行号，由 m = (b2b3b4b5)10 确定列号
    int row = expand[i] * 2 + expand[i + 5];
    int col = expand[i + 1] * 8 + expand[i + 2] * 4 + expand[i + 3] * 2 + expand[i + 4];
    int num = S_Boxs[i / 6][row][col];
    bitset<4> temp(num);    // 将十进制转为二进制
    // 按顺序连接成32位的串
    output[output_index] = temp[3];
    output[output_index + 1] = temp[2];
    output[output_index + 2] = temp[1];
    output[output_index + 3] = temp[0];
    output_index += 4;
}
```

最后，要对于连接好的 32 位 bit 进行 P 置换，得出轮函数的输出结果。

```
// P置换，输出32位
bitset<32> temp = output;
for (int i = 0; i < 32; ++i)
{
    output[i] = temp[P_Permutation_Table[i] - 1];
}
return output;
```

2. 对于生成子密钥的模块

首先对密钥 K 的 56 个非校验位实行置换 PC^{-1}，得到左右 28 位 bit

```
//PC-1置换
for (int i = 0; i < 56; ++i)
{
    not_checkout[i] = K[PC1_Permutation_Table[i] - 1];
}
for(int j = 0; j < 16; j++){
    // 实行前后28位分开
    for (int i = 0; i < 28; ++i)
    {
        C0[i] = not_checkout[i];
    }
    for (int i = 28; i < 56; ++i)
    {
        D0[i-28] = not_checkout[i];
    }
```

然后，进行左移操作，左移过程要注意第 1，2，9，16 只左移一位，其他位置左移两位。

```
// 左移操作
if(j == 1 || j == 2 || j == 9 || j == 16){
    C0 = leftshift(C0, 1);
    D0 = leftshift(D0, 1);
}
else{
    C0 = leftshift(C0, 2);
    D0 = leftshift(D0, 2);
}
```

最后，合并左右两部分 bit，将结果进行 PC2 置换，作为一个子密钥

```
//PC-2置换
for (int i = 0; i < 28; ++i)
{
    not_checkout[i] = C0[i];
}
for (int i = 28; i < 56; ++i)
{
    not_checkout[i] = D0[i - 28];
}
for (int i = 0; i < 48; ++i)
{
    subkey[i] = not_checkout[PC2_Permutation_Table[i] - 1];
}
sub_K[j] = subkey;
```

3. 对于 IP 置换模块，并分成左右两部分 bit

```
// IP置换
for (int i = 0; i < 64; ++i)
{
    after_IP_M[i] = M[IP_Permutation_Table[i] - 1];
}
for (int i = 0; i < 32; ++i)
{
    left[i] = after_IP_M[i];
}
for (int i = 32; i < 64; ++i)
{
    right[i - 32] = after_IP_M[i];
}
```

4. 对于 IP^{-1} 置换模块，置换完毕便为密文

```
// 逆置换IP-1
for (int i = 0; i < 64; ++i)
{
    cipher[i] = after_IP_M[IP_Inverse_Permutation_Table[i] - 1];
}
```

5. 字符串转 bitset，bitset 转字符串模块。

```
bitset<64> DES::char_to_bitset(const char s[8]){
    bitset<64> output;
    int output_index = 0;
    for (int i = 0; i < 8; ++i)
    {
        int num = int(s[i]);
        bitset<8> temp(num);
        for (int j = 0; j < 8; ++j)
        {
            output[output_index+j] = temp[7-j];
        }
        output_index += 8;
    }
    return output;
}
```

根据 bitset 中的二进制，每 8 位代表一个字符，首先将字符转换为 ASCII 码，然后再用二进制表示这个 ASCII 码

```
string DES::bitset_to_string(bitset<64> m){
    char s[9];
    memset(s,0,9);
    int s_index = 0;
    for(int i = 0; i < 64; i = i + 8){
        int temp = m[i]*128 + m[i+1]*64 + m[i+2]*32 + m[i+3]*16 + m[i+4]*8 + m[i+5]*4 + m[i+6]*2 + m[i+7];
        s[s_index++] = (char)temp;
    }
    string str = s;
    return str;
}
```

从二进制转换为字符也是同理，每八位进行处理，将解出的字符放入 string 中进行输出。

四. 数据结构

这里我使用了 c++库的 bitset 容器来表达 64 位明文以及密文，密钥等等，其中对输入的字符串明文进行了转换操作，同时对输出的解密结果也进行了转换，避免直接输出 64 位二进制数。

对于转换表，我主要采用了数组的形式，在进行转换表运算的过程要记得减去 1，才能正确的表达元素所在的位置。例如，下面的 IP 转换表，为 64 位的 int 类型数组。

```
int IP_Permutation_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
};
```

而对于 S 盒，我使用了一个三维的数组 S_Boxs[8][4][16]，第一位表达是取第几个盒子，第二位表达是该盒子的行数，第三位表达是该盒子的列数。

```
int S_Boxs[8][4][16] = {
    //box1
    {
        {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},
        {0, 15, 7, 4, 15, 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}
    },
    //box2
    {
        {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}
    },
    //box3
    {
```

五. C++语言源代码

由于代码行数较多，这里不一一展示，详细代码可见压缩包中 DES.hpp；DES.cpp; main.cpp.

其中 main.cpp 为程序运行的入口。按照提示输入相应指令即可实现 DES 加密解密功能。

六. 编译运行结果

```
********** Hint **********
Code:    encrypt——DES加密
         decrypt——DES解密
         exit——退出程序
**********************************
请输出指令:
encrypt
请输入明文字符串（最多8位char）以及密钥（8位char）:
hello!
12345678

DES加密过程:
明文:  hello!
密钥:  12345678
加密结果--64位密文:  0100110100011010100011010000000011101011100011110101101001110110
------------------------------------------------
```

输入明文：hello! 输入密钥：12345678

加密结果为 64 为密文：

0100110100011010100011010000000011101011100011110101101001110110

接着通过这个密文来解密出明文。

```
********** Hint **********
Code:    encrypt——DES加密
         decrypt——DES解密
         exit——退出程序
**********************************
请输出指令:
decrypt
请输入密文（64位bit）以及密钥（8位char）:
0100110100011010100011010000000011101011100011110101101001110110
12345678

DES解密过程:
64位密文:  0100110100011010100011010000000011101011100011110101101001110110
密钥:  12345678
解密结果--明文:  hello!
------------------------------------------------
```

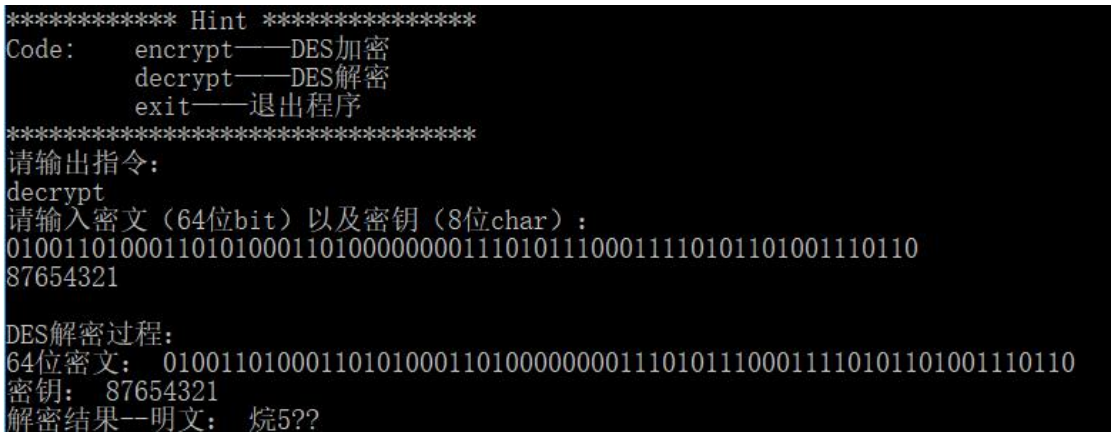
输入密文：上面的加密 64 位密文

输入密钥：12345678

解密结果为 hello!

由此可见该 DES 算法加密解密过程执行无误，实现正确！

接着来测试一下，当密钥错误，密文正确的情况，查看是否能解密



很明显解密结果出现的明文是乱码，即无法通过错误的密钥来解出明文

由此可见该 DES 算法加密解密过程是安全的，实现正确！