Assignment 1.

完成一个 DES 算法的详细设计，包括：

算法原理概述；总体结构；模块分解；数据结构；类-C 语言算法过程

DES 算法原理概述

DES（数据加密标准）是一种使用密钥加密的块密码，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 采用 64 位密钥，但由于每 8 位中的最后一位用于奇偶校验，因此实际有效密钥长度为 56 位。

设信息空间由{0,1}组成的字符串构成，明文信息和经过 DES 加密的密文信息是 64 位的分组，密钥也是 64 位。

明文：$M=m_1m_2...m_{64}$，$m_i\in\{0,1\}$，$i=1..64$

密文：$C=c_1c_2...c_{64}$，$c_i\in\{0,1\}$，$i=1..64$

密钥：$K=k_1k_2...k_{64}$，$k_i\in\{0,1\}$，$i=1..64$（除去 k_8，k_{16}，...，k_{64} 共 8 位奇偶校验位，起作用的仅为 56 位）

加密过程：

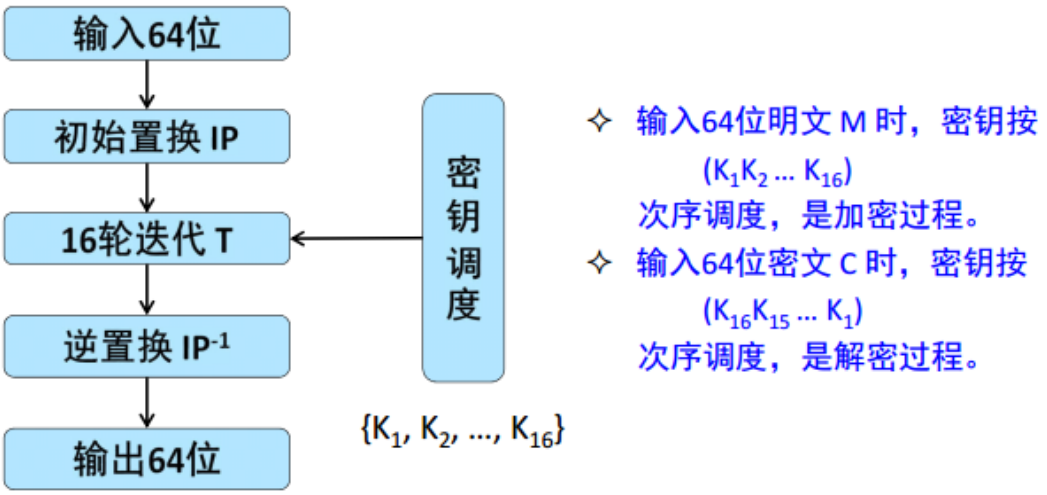
$C=E_k(M)=IP^{-1}\cdot T_{16}\cdot T_{15}\cdot...\cdot T_1\cdot IP(M)$，其中 IP 为初始置换，$IP^{-1}$ 是 IP 的逆，T_1，T_2，...，T_{16} 是一系列的迭代变换。

解密过程：

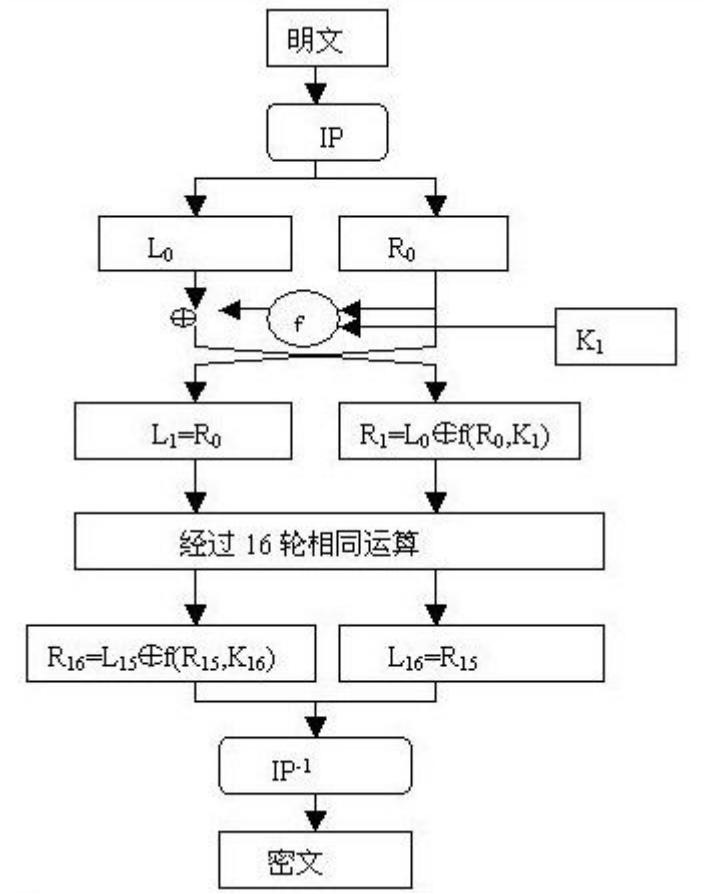
$$M=D_k(C)=IP^{-1}\cdot T_1\cdot T_2\cdot\ldots\cdot T_{16}\cdot IP(C)$$

DES 算法的总体结构

这是老师课件上的一个图：



对比一下网上的一张图：下面这张图的过程比较详细一点。



DES 的模块分解：

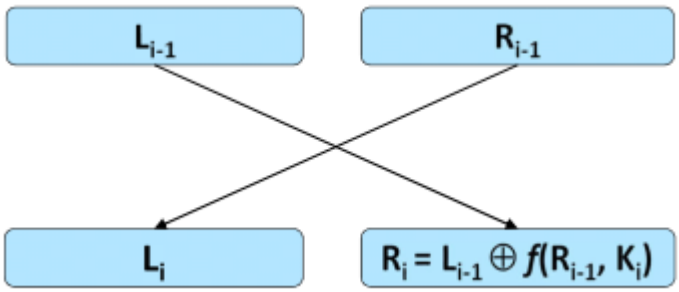
初始置换 IP：

给定 64 位明文块 M，通过一个固定的初始置换 IP 来重排 M 中的二进制位，得到二进制串 $M_0=IP(M)=L_0R_0$，这里 L_0 和 R_0 分别是 M_0 的前 32 位和后 32 位。下表给出 IP 置换后的下标编号序列：

IP 置换表							
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

表中的数字代表原数据中此位置的数据在新数据中的位置，即原数据的第 58 位放到新数据的第 1 位，第 50 位放到第 2 位，以此类推，第 7 位放到第 64 位。置换后的数据分为 L_0 和 R_0 两部分，L_0 为新数据的前 32 位，R_0 为新数据的后 32 位。

迭代 T：



根据 L_0R_0 按下列规则进行 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

这里是⊕32 位二进制串按位异或运算，f 是 Feistel 轮函数

16 个长度为 48bit 的子密钥 K_i(i=1...16)由密钥 K 生成；16 次迭代后得到 $L_{16}R_{16}$; 左右交换输出 $R_{16}L_{16}$

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：$C=IP^{-1}(R_{16}L_{16})$

IP 置换表							
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							
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

Feistel 轮函数 $f(R_{i-1}, K_i)$：

（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 K_i 作 48 位二进制串按位异或运算，K_i 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），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）得到的分组结果合并得到长度为 32 位的串

（5）将（4）的结果经过 P-置换，得到轮函数 $f(R_{i-1}, K_i)$的最终结果

E-扩展：

置换目标是 IP 置换后获得的右半部分 R_0，将 32 位输入扩展为 48 位（分为 4 位，8 组）输出；

置换目的有两个：生成与密钥相同长度的数据以进行异或运算；提供更长的结果，在后续的替代运算中可以进行压缩。

E-扩展规则 (比特-选择表)					
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

表中的数字代表位，左右两列数据是扩展的数据，扩展的数据是从相邻两组分别取靠近的一位，因此 4 位变为 6 位。

S-盒：

✧ S-盒

S_1-BOX															
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

S_2-BOX															
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

S_3-BOX															
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

S_4-BOX															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
12	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

✧ S-盒

S_5-BOX															
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

S_6-BOX															
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

S_7-BOX															
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

S_8-BOX															
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

S-盒是一类选择函数，用于二进制 6-4 转换。Feistel 轮函数使用 8 个 S-盒 S_1...S_8，每个 S-盒是一个 4 行（编号 0-3）、16 列（编号 0-15）的表，表中元素是一个 4 位二进制数的十进制表示，取值在 0-15 之间。

设 S_i 的 6 位输入为 $b_1b_2b_3b_4b_5b_6$，则由 $n=(b_1b_6)_{10}$ 确定行号，$m=(b_2b_3b_4b_5)$确定列号，$[S_i]_{n,m}$ 元素的值的二进制形式即为所要的 S_i 的输出。

P-置换：

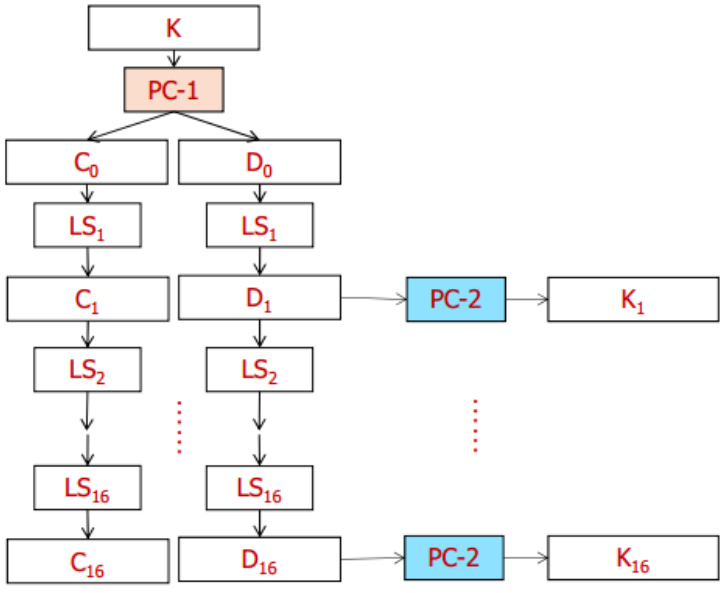
S 盒代替运算的 32 位输出按照 P 盒进行置换。该置换把输入的每位映射到输出位，任何一位不能被映射两次，也不能被略去。

P-置换表			
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

表中的数字代表原数据中此位置的数据在新数据中的位置，即原数据块的第 16 位放到新数据的第 1 位，第 7 位放到第 2 位，以此类推，第 25 位放到第 32 位。

最后，P 盒置换的结果与最初的 64 位分组左半部分 L_0 异或，然后左、右半部分交换，接着开始另一轮。

子密钥生成：



根据给定的 64 位密钥 K 生成 Feistel 轮函数的每轮中使用的子密钥

K_i。

（1）对 K 的 56 个非校验位实行置换 PC-1，得到 C_0D_0，其中 C_0 和 D_0 分别由 PC-1 置换后的前 28 位和后 28 位组成。

PC-1 置换表

C_0	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
D_0	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

（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 循环左移一位，否则循环左移两位。

（3）对 56 位的 C_iD_i 实行 PC-2 压缩置换，得到 48 位的 K_i

PC-2 压缩置换：从 56 位的 C_iD_i 中去掉第 9,18,22,25,35,38,43,54 位，将剩下的 48 位按照 PC-2 置换表作置换，得到 K_i

PC-2 压缩置换表

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

（4）如果已经得到 K_{16}，密钥调度过程结束，否则转（2）

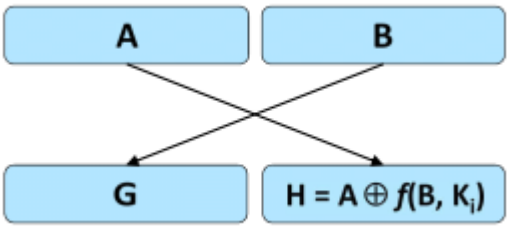
DES 的解密：

与加密过程不同之处在于子密钥的调度次序恰好相反。

加密过程的子密钥按{$K_1K_2...K_{15}K_{16}$}次序调度

解密过程的子密钥按{$K_{16}K_{15}...K_2K_1$}次序调度

过程如下：



64 位密文 C 输入 DES 过程，IP 置换后得到加密过程中的 $R_{16}L_{16}$；

对 $R_{16}L_{16}$ 实行 16 轮迭代，过程中 Feistel 轮函数按照相反次序引用子密钥 $K_{16}K_{15}...K_2K_1$

$A=R_{16}$，$B=L_{16}$

$G=B=L_{16}=R_{15}$

$H=A⊕f(B, K_{16})=R_{16}⊕f(L_{16}, K_{16})=L_{15}⊕f(R_{15}, K_{16})⊕f(R_{15}, K_{16})=L_{15}$

16 轮迭代结束时 $G=R_0$，$H=L_0$。左右交换得到 L_0R_0，即为加密过程中的 M_0

M_0 经过 IP^{-1} 置换得到原始明文 M，解密过程结束。

数据结构

数据结构主要是各种置换表，我们用数组的形式表示出来即可。

```
/* Initial Permutation Table */
static char 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
};

/* Inverse Initial Permutation Table */
static char PI[] = {
    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
};
```

```
/* Expansion table */
static char 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
};

/* Post S-Box permutation */
static char 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
};
```

```
/* The S-Box tables */
static char S[8][64] = {{
    /* 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,  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
},{
    /* S7 */
     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
},{
    /* 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
}};
```

```
/* Permuted Choice 1 Table */
static char PC1[] = {
    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
};

/* Permuted Choice 2 Table */
static char PC2[] = {
    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
};

/* Iteration Shift Array */
static char iteration_shift[] = {
 /* 1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16 */
    1,  1,  2,  2,  2,  2,  2,  2,  1,  2,  2,  2,  2,  2,  2,  1
};
```

类-C 语言实现 DES

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <stdint.h>
4
5   #define LB32_MASK 0x00000001
6   #define LB64_MASK 0x0000000000000001
7   #define L64_MASK 0x00000000ffffffff
8   #define H64_MASK 0xffffffff00000000
```

```c
/*
 * The DES function
 * input: 64 bit message
 * key: 64 bit key for encryption/decryption
 * mode: 'e' = encryption; 'd' = decryption
 */
uint64_t des(uint64_t input, uint64_t key, char mode) {
    int i, j;

    /* 8 bits */
    char row, column;

    /* 28 bits */
    uint32_t C = 0;
    uint32_t D = 0;

    /* 32 bits */
    uint32_t L = 0;
    uint32_t R = 0;
    uint32_t s_output = 0;
    uint32_t f_function_res = 0;
    uint32_t temp = 0;

    /* 48 bits */
    uint64_t sub_key[16] = {0};
    uint64_t s_input = 0;

    /* 56 bits */
    uint64_t permuted_choice_1 = 0;
    uint64_t permuted_choice_2 = 0;

    /* 64 bits */
    uint64_t init_perm_res = 0;
    uint64_t inv_init_perm_res = 0;
    uint64_t pre_output = 0;

```

```
176     /* initial permutation */
177     for (i = 0; i < 64; i++) {
178         init_perm_res <<= 1;
179         init_perm_res |= (input >> (64-IP[i])) & LB64_MASK;
180     }
181
182     L = (uint32_t)(init_perm_res >> 32) & L64_MASK;
183     R = (uint32_t)init_perm_res & L64_MASK;
184
185     /* initial key schedule calculation */
186     for (i = 0; i < 56; i++) {
187
188         permuted_choice_1 <<= 1;
189         permuted_choice_1 |= (key >> (64-PC1[i])) & LB64_MASK;
190
191     }
192
193     C = (uint32_t)((permuted_choice_1 >> 28) & 0x000000000fffffff);
194     D = (uint32_t)(permuted_choice_1 & 0x000000000fffffff);
195
```

```
196     /* Calculation of the 16 keys */
197     for (i = 0; i< 16; i++) {
198         /* key schedule */
199         // shifting Ci and Di
200         for (j = 0; j < iteration_shift[i]; j++) {
201             C = 0x0fffffff & (C << 1) | 0x00000001 & (C >> 27);
202             D = 0x0fffffff & (D << 1) | 0x00000001 & (D >> 27);
203         }
204
205         permuted_choice_2 = 0;
206         permuted_choice_2 = (((uint64_t)C) << 28) | (uint64_t)D;
207
208         sub_key[i] = 0;
209
210         for (j = 0; j < 48; j++) {
211             sub_key[i] <<= 1;
212             sub_key[i] |= (permuted_choice_2 >> (56-PC2[j])) & LB64_MASK;
213         }
214     }
215
```

```
216     for (i = 0; i < 16; i++) {
217         /* f(R,k) function */
218         s_input = 0;
219         for (j = 0; j< 48; j++) {
220             s_input <<= 1;
221             s_input |= (uint64_t) ((R >> (32-E[j])) & LB32_MASK);
222         }
223 
224         /*
225          * Encryption/Decryption
226          * XORing expanded Ri with Ki
227          */
228         if (mode == 'd') {
229             // decryption
230             s_input = s_input ^ sub_key[15-i];
231         } else {
232             // encryption
233             s_input = s_input ^ sub_key[i];
234         }
235 
```

```
236     /* S-Box Tables */
237     for (j = 0; j < 8; j++) {
238         // 00 00 RCCC CR00 00 00 00 00 00 s_input
239         // 00 00 1000 0100 00 00 00 00 00 row mask
240         // 00 00 0111 1000 00 00 00 00 00 column mask
241         row = (char) ((s_input & (0x0000840000000000 >> 6*j)) >> 42-6*j);
242         row = (row >> 4) | row & 0x01;
243 
244         column = (char) ((s_input & (0x0000780000000000 >> 6*j)) >> 43-6*j);
245 
246         s_output <<= 4;
247         s_output |= (uint32_t) (S[j][16*row + column] & 0x0f);
248     }
```

```
250         f_function_res = 0;
251         for (j = 0; j < 32; j++) {
252             f_function_res <<= 1;
253             f_function_res |= (s_output >> (32 - P[j])) & LB32_MASK;
254         }
255         temp = R;
256         R = L ^ f_function_res;
257         L = temp;
258     }
259     pre_output = (((uint64_t) R) << 32) | (uint64_t) L;
260     /* inverse initial permutation */
261     for (i = 0; i < 64; i++) {
262         inv_init_perm_res <<= 1;
263         inv_init_perm_res |= (pre_output >> (64-PI[i])) & LB64_MASK;
264     }
265     return inv_init_perm_res;
266 }
```