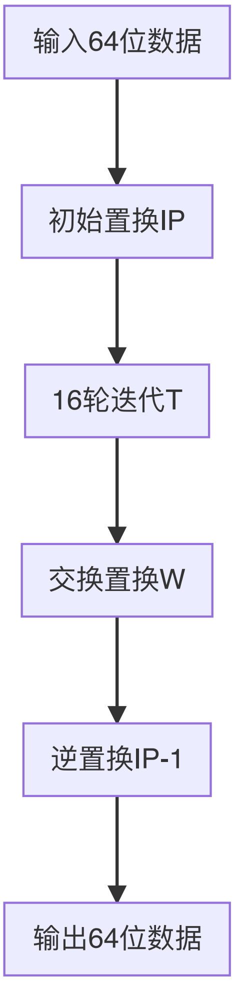
信息安全课程作业1

完成一个DES算法的程序设计和实现，包括

- 算法原理概述；总体结构；模块分解；数据结构设计；C语言源代码；编译运行结果

总体结构

DES以64位为分组对数据加密，加密和解密用的是同一个算法。密钥长64位，但其中56位参与DES运算，第8、16、24、32、40、48、56、64是校验位，使得每个密钥都有奇数个1，分组后的明文组和56位的密钥按位替代或交换的方式形成密文组。DES算法的主要流程大致如下：



> 1. 对64位明文进行初始置换
> 2. 将IP置换后的64位数据分为两部分，左32位记为L，右32位记为R
> 3. 去掉奇偶校验位的密钥为56位（PC1），将其左右分为28位，分别将左右28位进行循环左移位（rotateLeft）再合并为56位，再经压缩（PC2）后为48位子密钥（geneSubKey）
> 4. 对R进行扩展置换（E），得到48位的数据与密钥压缩后的48位异或
> 5. 将经过异或后的48位数据分为8组，每组6位放入S盒中，输出8组4位，总共为32位

> （sBox），再经过（P）
> 6. 将上面得到的32位数据与L异或结果赋给R
> 7. 前15次依次交换L和R位置
> 8. 重复以上2-7过程16次
> 9. 将最后的到的L和R合并做逆置换（invIP）

模块分解

加密过程：

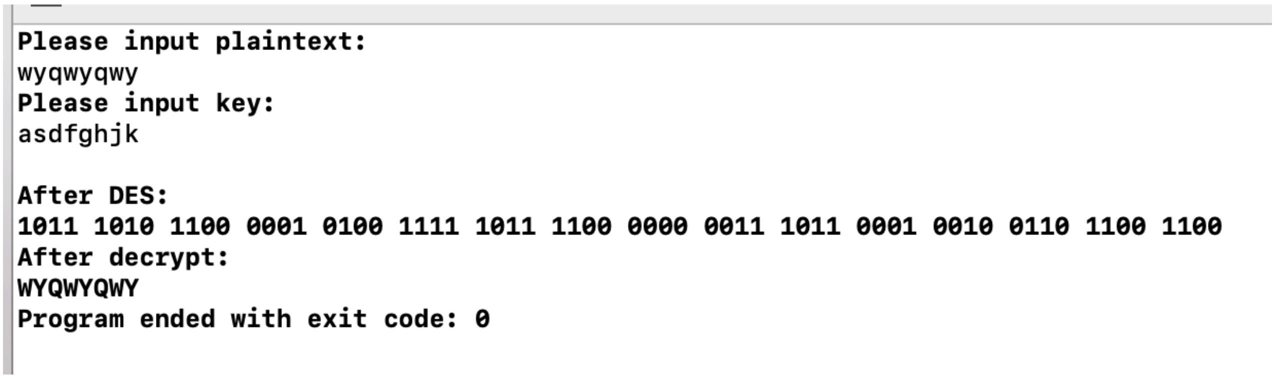
- 初始置换
- 16轮迭代
  - Feistel轮函数
    - E扩展
    - 子密钥的生成
    - S盒
    - P置换
- 左右置换
- 逆置换

数据结构设计

我使用的数据结构都相对简单：

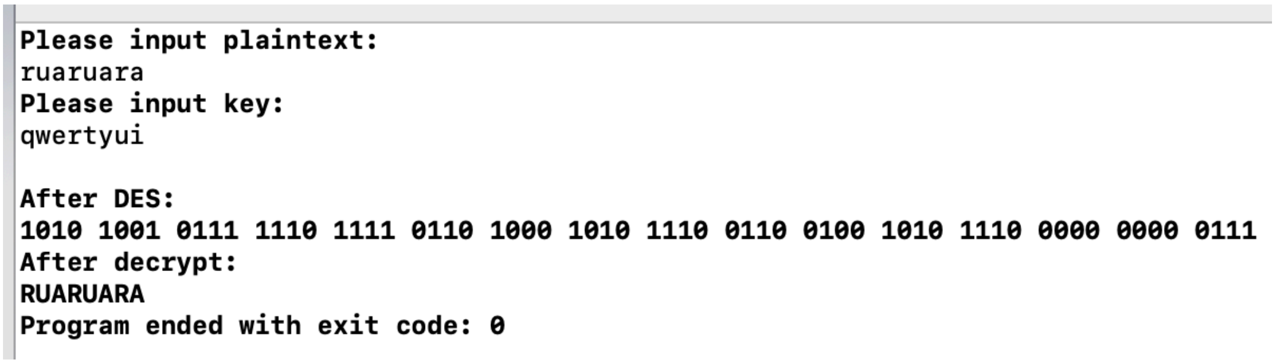
- 输入明文和密钥都使用`char []`存储（使用时的转换通过实现char2int和int2char函数来完成）
- 各置换表使用`int[]`数组存储，s盒则使用3维数组存储

编译运行结果



```
Please input plaintext:
wyqwyqwy
Please input key:
asdfghjk

After DES:
1011 1010 1100 0001 0100 1111 1011 1100 0000 0011 1011 0001 0010 0110 1100 1100
After decrypt:
WYQWYQWY
Program ended with exit code: 0
```



```
Please input plaintext:
ruaruara
Please input key:
qwertyui

After DES:
1010 1001 0111 1110 1111 0110 1000 1010 1110 0110 0100 1010 1110 0000 0000 0111
After decrypt:
RUARUARA
Program ended with exit code: 0
```

源码

（实现的相对简单，所以只支持8位的输入，没有实现不足64补齐的操作）

```
#include<stdio.h>
#include<string.h>

// 8 sbox
int sBox[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,  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
};


int ipMatrix[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
};

int eMatrix[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
};

int pBox[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
};

int invIpMatrix[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
};

int pcMatrix1[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
};
```

```
int pcMatrix2[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
};

// char和bit转换
static void char2Bit(const char input[], int output[], int bits){
    for (int j = 0; j<8; j++){
        for (int i = 0; i<8; i++)
            output[7 * (j + 1) - i + j] = (input[j] >> i) & 1;
    }
};

static void bit2Char(const int intput[], char output[], int bits){
    for (int j = 0; j < 8; j++){
        for (int i = 0; i<8; i++)
            output[j] = output[j] * 2 + intput[i + 8 * j];
    }
};

// 初始IP置换
static  void IP(const int input[64], int output[64], int table[64]){
    for (int i = 0; i < 64; i++)
        output[i] = input[table[i] - 1];

};

// E扩展
static  void E(const int input[32], int output[48], int table[48]){
    for (int i = 0; i < 48; i++)
        output[i] = input[table[i] - 1];

};

// 异或
static void Xor(int *input1, int *input2, int len){
    for (int i = 0; i < len; i++)
        *(input1 + i) = *(input1 + i) ^ *(input2 + i);

};

// S盒
static  void S(const int input[48], int output[32], int table[8][4][16]){
    int i = 0;
    int j = 0;
    int INT[8];
    for (; i<48; i = i + 6){
        INT[j] = table[j][(input[i] << 1) + (input[i + 5])][(input[i + 1] << 3)
+ (input[i + 2] << 2) + (input[i + 3] << 1) + (input[i + 4])];
        j++;
    }
    for (j = 0; j < 8; j++){
        for (i = 0; i < 4; i++)
            output[3 * (j + 1) - i + j] = (INT[j] >> i) & 1;

    }
};


// P置换
static  void P(const int input[32], int output[32], int table[32]){
    for (int i = 0; i < 32; i++)
        output[i] = input[table[i] - 1];

};

// 密钥相关
// 逆IP
static  void invIP(const int input[64], int output[64], int table[64]){
    for (int i = 0; i < 64; i++)
        output[i] = input[table[i] - 1];

};


static  void PC_1(const int input[64], int output[56], int table[56]){
    for (int i = 0; i < 56; i++)
        output[i] = input[table[i] - 1];

};

// 秘钥循环左移
static void rotateLeft(const int input[28], int output[28], int leftCount){
    int len = 28;
    for (int i = 0; i < len; i++)
        output[i] = input[(i + leftCount) % len];

};


// PC_2
static  void PC_2(const int input[56], int output[48], int table[48]){
    for (int i = 0; i < 48; i++)
        output[i] = input[table[i] - 1];

};
```

```
// 轮变换
static void F_func(int input[32], int output[32], int subkey[48]){
    int len = 48;
    int temp[48] = { 0 };
    int temp_1[32] = { 0 };
    E(input, temp, eMatrix);
    Xor(temp, subkey, len);
    S(temp, temp_1, sBox);
    P(temp_1, output, pBox);
};

// 生成子密钥
static void  geneSubkey(const int input[64], int Subkey[4][48]){
    int loop = 1, loop_2 = 2;
    int i, j;
    int c[28], d[28];
    int pc_1[56] = { 0 };
    int pc_2[4][56] = { 0 };
    int rotatel_c[4][28] = { 0 };
    int rotatel_d[4][28] = { 0 };
    PC_1(input, pc_1, pcMatrix1);
    for (i = 0; i < 28; i++){
        c[i] = pc_1[i];
        d[i] = pc_1[i + 28];
    }
    int leftCount = 0;
    for (i = 1; i < 5; i++){
        if (i == 1 || i == 2 || i == 9 || i == 16){
            leftCount += loop;
            rotateLeft(c, rotatel_c[i - 1], leftCount);
            rotateLeft(d, rotatel_d[i - 1], leftCount);
        }else{
            leftCount += loop_2;
            rotateLeft(c, rotatel_c[i - 1], leftCount);
            rotateLeft(d, rotatel_d[i - 1], leftCount);
        }
    }
    for (i = 0; i < 4; i++){
        for (j = 0; j < 28; j++){
            pc_2[i][j] = rotatel_c[i][j];
            pc_2[i][j + 28] = rotatel_d[i][j];
        }
    }
    for (i = 0; i < 4; i++){
        PC_2(pc_2[i], Subkey[i], pcMatrix2);
    }
};
```

```
static void  encrypt(char input[8], char key_in[8], int output[64]){
    int afterInit[64] = { 0 };
    int output_1[64] = { 0 };
    int subkeys[4][48];
    int chartobit[64] = { 0 };
    int key[64];
    int l[5][32], r[5][32];
    char2Bit(input, chartobit, 8);
    IP(chartobit, afterInit, ipMatrix);
    char2Bit(key_in, key, 8);
    geneSubkey(key, subkeys);

    for (int i = 0; i<32; i++){
        l[0][i] = afterInit[i];
        r[0][i] = afterInit[32 + i];
    }
    //这里我做了四轮
    for (int j = 1; j<4; j++){
        for (int k = 0; k<32; k++)
            l[j][k] = r[j - 1][k];

        F_func(r[j - 1], r[j], subkeys[j - 1]);
        Xor(r[j], l[j - 1], 32);
    }
    int t = 0;
    for (t = 0; t<32; t++)
        r[4][t] = r[3][t];

    F_func(r[3], l[4], subkeys[3]);
    Xor(l[4], l[3], 32);

    // 合并
    for (t = 0; t<32; t++)  {
        output_1[t] = l[4][t];
        output_1[32 + t] = r[4][t];
    }

    invIP(output_1, output, invIpMatrix);
};

static void  decrypt(int input[64], char key_in[8], char output[8]){
    int Ip[64] = { 0 };
    int output_1[64] = { 0 };
    int output_2[64] = { 0 };
    int subkeys[4][48];
    int key[64];
    int l[5][32], r[5][32];
    IP(input, Ip, ipMatrix);
    char2Bit(key_in, key, 8);
    geneSubkey(key, subkeys);
    for (int i = 0; i < 32; i++){
        l[0][i] = Ip[i];
        r[0][i] = Ip[32 + i];
    }

    for (int j = 1; j<4; j++){
        for (int k = 0; k<32; k++)
            l[j][k] = r[j - 1][k];

        F_func(r[j - 1], r[j], subkeys[4 - j]);
        Xor(r[j], l[j - 1], 32);
    }
    int t = 0;
    for (t = 0; t<32; t++)
        r[4][t] = r[3][t];

    F_func(r[3], l[4], subkeys[0]);
    Xor(l[4], l[3], 32);

    for (t = 0; t<32; t++){
        output_1[t] = l[4][t];
        output_1[32 + t] = r[4][t];
    }

    invIP(output_1, output_2, invIpMatrix);
    bit2Char(output_2, output, 8);
};

int main(){
    int output[64] = { 0 };
    char plaintext[9] = { 0 };
    char keys[9] = { 0 };
    printf("Please input plaintext: \n");
    fgets(plaintext, (sizeof(plaintext) / sizeof plaintext[0]), stdin);
    fflush(stdin);
    printf("Please input key: \n");
    fgets(keys, (sizeof keys / sizeof keys[0]), stdin);
    encrypt(plaintext, keys, output);

    printf("\nAfter DES:\n");
    for (int i = 0; i<64; i++){
        printf("%d", output[i]);
        if ((i + 1) % 4 == 0)
            printf(" ");
    }
    printf("\n");
    decrypt(output, keys, plaintext);
    printf("After decrypt:\n");
    for (int i = 0; i<8; i++)
        printf("%c", plaintext[i]);

    printf("\n");
    return 0;
}
```