AES加密算法c语言实现代码

```
#include "stdio.h"
#include "memory.h"
#include "time.h"
#include "stdlib.h"

#define PLAIN_FILE_OPEN_ERROR -1
#define KEY_FILE_OPEN_ERROR -2
#define CIPHER_FILE_OPEN_ERROR -3
#define OK 1

typedef char ElemType;

 /*初始置换表IP*/
int IP_Table[64] = {  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,
56,48,40,32,24,16,8,0,
58,50,42,34,26,18,10,2,
60,52,44,36,28,20,12,4,
62,54,46,38,30,22,14,6};
/*逆初始置换表IP^-1*/
int IP_1_Table[64] = {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,
  32,0,40,8,48,16,56,24};

/*扩充置换表E*/
int E_Table[48] = {31,  0,  1,  2,  3,  4,
3,   4,  5,  6,  7,  8,
7,   8,9,10,11,12,
11,12,13,14,15,16,
15,16,17,18,19,20,
19,20,21,22,23,24,
23,24,25,26,27,28,
27,28,29,30,31,  0};

/*置换函数P*/
int P_Table[32] = {15,6,19,20,28,11,27,16,
0,14,22,25,4,17,30,9,
1,7,23,13,31,26,2,8,
18,12,29,5,21,10,3,24};

/*S盒*/
int S[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}}};
/*置换选择1*/
int PC_1[56] = {56,48,40,32,24,16,8,
  0,57,49,41,33,25,17,
  9,1,58,50,42,34,26,
  18,10,2,59,51,43,35,
  62,54,46,38,30,22,14,
  6,61,53,45,37,29,21,
  13,5,60,52,44,36,28,
  20,12,4,27,19,11,3};

/*置换选择2*/
int PC_2[48] = {13,16,10,23,0,4,2,27,
  14,5,20,9,22,18,11,3,
  25,7,15,6,26,19,12,1,
  40,51,30,36,46,54,29,39,
  50,44,32,46,43,48,38,55,
  33,52,45,41,49,35,28,31};

/*对左移次数的规定*/
int MOVE_TIMES[16] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

int ByteToBit(ElemType ch,ElemType bit[8]);
int BitToByte(ElemType bit[8],ElemType *ch);
int Char8ToBit64(ElemType ch[8],ElemType bit[64]);
int Bit64ToChar8(ElemType bit[64],ElemType ch[8]);
int DES_MakeSubKeys(ElemType key[64],ElemType subKeys[16][48]);
int DES_PC1_Transform(ElemType key[64], ElemType tempbts[56]);
int DES_PC2_Transform(ElemType key[56], ElemType tempbts[48]);
int DES_ROL(ElemType data[56], int time);
int DES_IP_Transform(ElemType data[64]);
int DES_IP_1_Transform(ElemType data[64]);
int DES_E_Transform(ElemType data[48]);
int DES_P_Transform(ElemType data[32]);
int DES_SBOX(ElemType data[48]);
int DES_XOR(ElemType R[48], ElemType L[48],int count);
int DES_Swap(ElemType left[32],ElemType right[32]);
int DES_EncryptBlock(ElemType plainBlock[8], ElemType subKeys[16][48], ElemType cipherBlock[8]);
int DES_DecryptBlock(ElemType cipherBlock[8], ElemType subKeys[16][48], ElemType plainBlock[8]);
int DES_Encrypt(char *plainFile, char *keyStr,char *cipherFile);
int DES_Decrypt(char *cipherFile, char *keyStr,char *plainFile);

/*字节转换成二进制*/
int ByteToBit(ElemType ch, ElemType bit[8]){
 int cnt;
 for(cnt = 0;cnt < 8; cnt++){
  *(bit+cnt) = (ch>>cnt)&1;
 }
 return 0;
}

/*二进制转换成字节*/
int BitToByte(ElemType bit[8],ElemType *ch){
 int cnt;
 for(cnt = 0;cnt < 8; cnt++){
  *ch |= *(bit + cnt)<<cnt;
 }
 return 0;
}

/*将长度为8的字符串转为二进制位串*/
int Char8ToBit64(ElemType ch[8],ElemType bit[64]){
```

```
 int cnt;
 for(cnt = 0; cnt < 8; cnt++){
  ByteToBit(*(ch+cnt),bit+(cnt<<3));
 }
 return 0;
}

/*将二进制位串转为长度为8的字符串*/
int Bit64ToChar8(ElemType bit[64],ElemType ch[8]){
 int cnt;
 memset(ch,0,8);
 for(cnt = 0; cnt < 8; cnt++){
  BitToByte(bit+(cnt<<3),ch+cnt);
 }
 return 0;
}

/*生成子密钥*/
int DES_MakeSubKeys(ElemType key[64],ElemType subKeys[16][48]){
 ElemType temp[56];
 int cnt;
 DES_PC1_Transform(key,temp);/*PC1置换*/
 for(cnt = 0; cnt < 16; cnt++){/*16轮跌代，产生16个子密钥*/
  DES_ROL(temp,MOVE_TIMES[cnt]);/*循环左移*/
  DES_PC2_Transform(temp,subKeys[cnt]);/*PC2置换，产生子密钥*/
 }
 return 0;
}

/*密钥置换1*/
int DES_PC1_Transform(ElemType key[64], ElemType tempbts[56]){
 int cnt;
 for(cnt = 0; cnt < 56; cnt++){
  tempbts[cnt] = key[PC_1[cnt]];
 }
 return 0;
}

/*密钥置换2*/
int DES_PC2_Transform(ElemType key[56], ElemType tempbts[48]){
 int cnt;
 for(cnt = 0; cnt < 48; cnt++){
  tempbts[cnt] = key[PC_2[cnt]];
 }
 return 0;
}

/*循环左移*/
int DES_ROL(ElemType data[56], int time){
 ElemType temp[56];

 /*保存将要循环移动到右边的位*/
 memcpy(temp,data,time);
 memcpy(temp+time,data+28,time);

 /*前28位移动*/
 memcpy(data,data+time,28-time);
 memcpy(data+28-time,temp,time);

 /*后28位移动*/
 memcpy(data+28,data+28+time,28-time);
 memcpy(data+56-time,temp+time,time);

 return 0;
}

/*IP置换*/
int DES_IP_Transform(ElemType data[64]){
 int cnt;
 ElemType temp[64];
 for(cnt = 0; cnt < 64; cnt++){
  temp[cnt] = data[IP_Table[cnt]];
 }
 memcpy(data,temp,64);
 return 0;
```

```
}

/*IP逆置换*/
int DES_IP_1_Transform(ElemType data[64]){
 int cnt;
 ElemType temp[64];
 for(cnt = 0; cnt < 64; cnt++){
  temp[cnt] = data[IP_1_Table[cnt]];
 }
 memcpy(data,temp,64);
 return 0;
}

/*扩展置换*/
int DES_E_Transform(ElemType data[48]){
 int cnt;
 ElemType temp[48];
 for(cnt = 0; cnt < 48; cnt++){
  temp[cnt] = data[E_Table[cnt]];
 }
 memcpy(data,temp,48);
 return 0;
}

/*P置换*/
int DES_P_Transform(ElemType data[32]){
 int cnt;
 ElemType temp[32];
 for(cnt = 0; cnt < 32; cnt++){
  temp[cnt] = data[P_Table[cnt]];
 }
 memcpy(data,temp,32);
 return 0;
}

/*异或*/
int DES_XOR(ElemType R[48], ElemType L[48] ,int count){
 int cnt;
 for(cnt = 0; cnt < count; cnt++){
  R[cnt] ^= L[cnt];
 }
 return 0;
}

/*S盒置换*/
int DES_SBOX(ElemType data[48]){
 int cnt;
 int line,row,output;
 int cur1,cur2;
 for(cnt = 0; cnt < 8; cnt++){
  cur1 = cnt*6;
  cur2 = cnt<<2;

  /*计算在S盒中的行与列*/
  line = (data[cur1]<<1) + data[cur1+5];
  row = (data[cur1+1]<<3) + (data[cur1+2]<<2)
+ (data[cur1+3]<<1) + data[cur1+4];
  output = S[cnt][line][row];

  /*化为2进制*/
  data[cur2] = (output&0X08)>>3;
  data[cur2+1] = (output&0X04)>>2;
  data[cur2+2] = (output&0X02)>>1;
  data[cur2+3] = output&0x01;
 }
 return 0;
}

/*交换*/
int DES_Swap(ElemType left[32], ElemType right[32]){
 ElemType temp[32];
 memcpy(temp,left,32);
 memcpy(left,right,32);
 memcpy(right,temp,32);
 return 0;
```

```
}

/*加密单个分组*/
int DES_EncryptBlock(ElemType plainBlock[8], ElemType subKeys[16][48], ElemType cipherBlock[8]){
 ElemType plainBits[64];
 ElemType copyRight[48];
 int cnt;

 Char8ToBit64(plainBlock,plainBits);
 /*初始置换（IP置换）*/
 DES_IP_Transform(plainBits);

 /*16轮迭代*/
 for(cnt = 0; cnt < 16; cnt++){
  memcpy(copyRight,plainBits+32,32);
  /*将右半部分进行扩展置换，从32位扩展到48位*/
  DES_E_Transform(copyRight);
  /*将右半部分与子密钥进行异或操作*/
  DES_XOR(copyRight,subKeys[cnt],48);
  /*异或结果进入S盒，输出32位结果*/
  DES_SBOX(copyRight);
  /*P置换*/
  DES_P_Transform(copyRight);
  /*将明文左半部分与右半部分进行异或*/
  DES_XOR(plainBits,copyRight,32);
  if(cnt != 15){
/*最终完成左右部的交换*/
DES_Swap(plainBits,plainBits+32);
  }
 }
 /*逆初始置换（IP^1置换）*/
 DES_IP_1_Transform(plainBits);
 Bit64ToChar8(plainBits,cipherBlock);
 return 0;
}

/*解密单个分组*/
int DES_DecryptBlock(ElemType cipherBlock[8], ElemType subKeys[16][48],ElemType plainBlock[8]){
 ElemType cipherBits[64];
 ElemType copyRight[48];
 int cnt;

 Char8ToBit64(cipherBlock,cipherBits);
 /*初始置换（IP置换）*/
 DES_IP_Transform(cipherBits);

 /*16轮迭代*/
 for(cnt = 15; cnt >= 0; cnt--){
  memcpy(copyRight,cipherBits+32,32);
  /*将右半部分进行扩展置换，从32位扩展到48位*/
  DES_E_Transform(copyRight);
  /*将右半部分与子密钥进行异或操作*/
  DES_XOR(copyRight,subKeys[cnt],48);
  /*异或结果进入S盒，输出32位结果*/
  DES_SBOX(copyRight);
  /*P置换*/
  DES_P_Transform(copyRight);
  /*将明文左半部分与右半部分进行异或*/
  DES_XOR(cipherBits,copyRight,32);
  if(cnt != 0){
/*最终完成左右部的交换*/
DES_Swap(cipherBits,cipherBits+32);
  }
 }
 /*逆初始置换（IP^1置换）*/
 DES_IP_1_Transform(cipherBits);
 Bit64ToChar8(cipherBits,plainBlock);
 return 0;
}

/*加密文件*/
int DES_Encrypt(char *plainFile, char *keyStr,char *cipherFile){
 FILE *plain,*cipher;
 int count;
 ElemType plainBlock[8],cipherBlock[8],keyBlock[8];
```

```
 ElemType bKey[64];
 ElemType subKeys[16][48];
 if((plain = fopen(plainFile,"rb")) == NULL){
  return PLAIN_FILE_OPEN_ERROR;
 }
 if((cipher = fopen(cipherFile,"wb")) == NULL){
  return CIPHER_FILE_OPEN_ERROR;
 }
 /*设置密钥*/
 memcpy(keyBlock,keyStr,8);
 /*将密钥转换为二进制流*/
 Char8ToBit64(keyBlock,bKey);
 /*生成子密钥*/
 DES_MakeSubKeys(bKey,subKeys);

 while(!feof(plain)){
  /*每次读8个字节，并返回成功读取的字节数*/
  if((count = fread(plainBlock,sizeof(char),8,plain)) == 8){
DES_EncryptBlock(plainBlock,subKeys,cipherBlock);
fwrite(cipherBlock,sizeof(char),8,cipher);
  }
 }
 if(count){
  /*填充*/
  memset(plainBlock + count,'\0',7 - count);
  /*最后一个字符保存包括最后一个字符在内的所填充的字符数量*/
  plainBlock[7] = 8 - count;
  DES_EncryptBlock(plainBlock,subKeys,cipherBlock);
  fwrite(cipherBlock,sizeof(char),8,cipher);
 }
 fclose(plain);
 fclose(cipher);
 return OK;
}

/*解密文件*/
int DES_Decrypt(char *cipherFile, char *keyStr,char *plainFile){
 FILE *plain, *cipher;
 int count,times = 0;
 long fileLen;
 ElemType plainBlock[8],cipherBlock[8],keyBlock[8];
 ElemType bKey[64];
 ElemType subKeys[16][48];
 if((cipher = fopen(cipherFile,"rb")) == NULL){
  return CIPHER_FILE_OPEN_ERROR;
 }
 if((plain = fopen(plainFile,"wb")) == NULL){
  return PLAIN_FILE_OPEN_ERROR;
 }

 /*设置密钥*/
 memcpy(keyBlock,keyStr,8);
 /*将密钥转换为二进制流*/
 Char8ToBit64(keyBlock,bKey);
 /*生成子密钥*/
 DES_MakeSubKeys(bKey,subKeys);

 /*取文件长度 */
 fseek(cipher,0,SEEK_END);/*将文件指针置尾*/
 fileLen = ftell(cipher); /*取文件指针当前位置*/
 rewind(cipher); /*将文件指针重指向文件头*/
 while(1){
  /*密文的字节数一定是8的整数倍*/
  fread(cipherBlock,sizeof(char),8,cipher);
  DES_DecryptBlock(cipherBlock,subKeys,plainBlock);
  times += 8;
  if(times < fileLen){
fwrite(plainBlock,sizeof(char),8,plain);
  }
  else{
break;
  }
 }
 /*判断末尾是否被填充*/
 if(plainBlock[7] < 8){
```

```
  for(count = 8 - plainBlock[7]; count < 7; count++){
if(plainBlock[count] != '\0'){
 break;
}
  }
 }
 if(count == 7){/*有填充*/
  fwrite(plainBlock,sizeof(char),8 - plainBlock[7],plain);
 }
 else{/*无填充*/
  fwrite(plainBlock,sizeof(char),8,plain);
 }

 fclose(plain);
 fclose(cipher);
 return OK;
}

int main()
{
 clock_t a,b;
 a = clock();
 DES_Encrypt("1.txt","key.txt","2.txt");
 b = clock();
 printf("加密消耗%d毫秒\n",b-a);

 system("pause");
 a = clock();
 DES_Decrypt("2.txt","key.txt","3.txt");
 b = clock();
 printf("解密消耗%d毫秒\n",b-a);
 getchar();
 return 0;
}
```