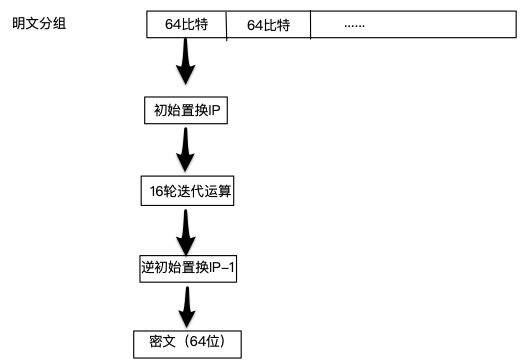
**算法原理概述：**

DES算法为密码体制中的对称密码体制，又被称为美国数据加密标准，是1972年美国IBM公司研制的对称密码体制加密算法，英文全称是Data Encryption Standard。于1973年5月被美国采纳为联邦信息处理标准。该标准每5年审查一次。因为DES的安全性出现问题，同时AES的出现，美联邦在1994年1月取消了DES作为联邦加密标准。DES加密不断被破解，其中用时最短的时间是22小时15分钟，所以DES算法现在应用越来越少了。  
DES是以64比特的明文为一个单位来进行加密的，超过64比特的数据，要求按固定的64比特的大小分组。每组64比特的明文加密得到同样长度的密文。DES的密钥长度为64位。加密运算时实际用到的密钥长度是56位，原密钥舍弃掉8位比特，分别是每隔8位的比特，即原密钥的第8位，第16位，......，第64位。而舍弃掉的这8位比特作用是校验奇偶性的。这8个比特的定义如下：若其前面7个比特中有奇数个1，则该比特为0，反之为1。

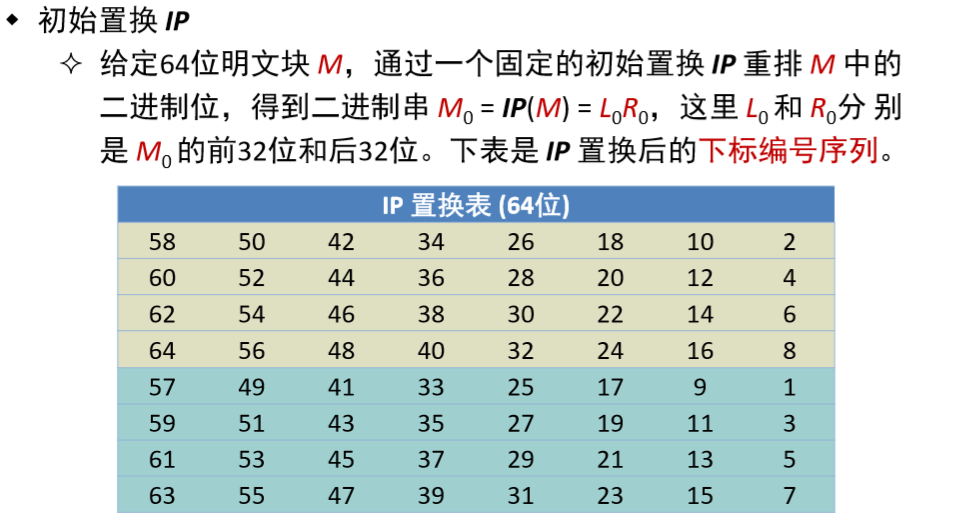
1. DES算法加密过程

DES是一个迭代分组密码，在对明文加密之前先对明文进行补长，使补长后明文的比特长度模64为0，再按照每组64比特分组。依次对分组密文进行加密，最终把加密后的结果拼接一起，得到密文。  
每组64位的输入数据块m的加密过程如下：  
1) 首先m经过初始置换IP得到m0 ；  
2) 将m0分成左右各为32比特两部分，记为m0 = L0 R0 ；  
3) 对L0和R0进行16轮迭代运算加密，得到L16和R16；  
4) 再对L16R16进行初始置换IP的逆初始置换IP^-1 ，得到该分组输入块的密文。  
DES加密总体框架如下图：  


1.1 初始置换IP

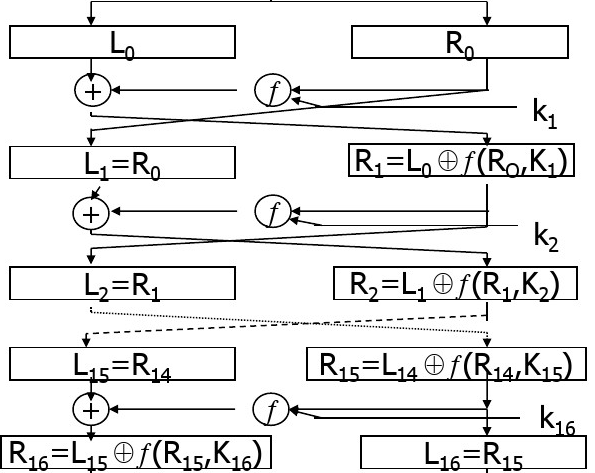
初始置换IP是将一个64比特的消息中的各个比特进行换位，目的是让消息中的各个比特的顺序错乱。设m=m1m2...m64，根据初始置换IP表进行置换。初始置换IP表里的元素代表m的第几位。置换原则是按行依次对m的每个比特进行替换，比如m1置换后是m58，m2置换后是m50，以此类推，最后的m64置换后是m7。经过置换后的分组明文块由两部分组成，记为L0R0。

初始置换IP表：



1.2 16轮迭代运算

迭代运算的结构是Feistel。在Feistel结构中，加密的每个过程称为轮，全过程就是若干轮的加密运算。DES第16轮加密运算与前15轮不一样，过程分为两步：1.和前15轮一样得到L16和R16；2.将得到的L16和R16两部分整体进行互换，得到最终的L16R16。  
每一轮的运算规则如下：  
Li = Ri-1  
Ri = Li-1 ⊕ f(Ri-1 , ki)  
其中L0和R0已知；⊕表示两个比特串按位异或，f是一个非线形函数，ki是由密钥按照一定规则每一轮生成的长度均为48位的比特串。  
具体的迭代过程如下图所示：



迭代过程中最重要的内容是每轮子密钥ki的生成和非线形函数f。下面我们依次分析这两部分内容。

1.2.1 子密钥ki的生成

子密钥的生成是随着f函数的轮数顺序产生的。生成过程如下：  
1） 给定一个64比特的初始密钥k，利用置换PC-1（表1.2.1）得到56位密钥，然后将这56位密钥分成左右两部分，前28比特记为C0，后28比特记为D0。经过这次置换就会丢弃掉8位比特，这在DES的开始部分已经有解释。  
2） 对于第i轮，1<=i<=16，首先计算  
Ci=LSi(Ci-1)  
Di=LSi(Di-1)  
其中，LSi表示左循环移位，当i=1，2，9，16时，左循环移1位，当i=3，4，5，6，7，8，10，11，12，13，14，15时，左循环移位2位。  
经过上边的左移位后，我们会得到当前i轮的Ci和Di。  
3） 将移位后的Ci和Di利用PC-2（表1.2.2）置换，最终得到长度为48位的比特串，也就是子密钥。

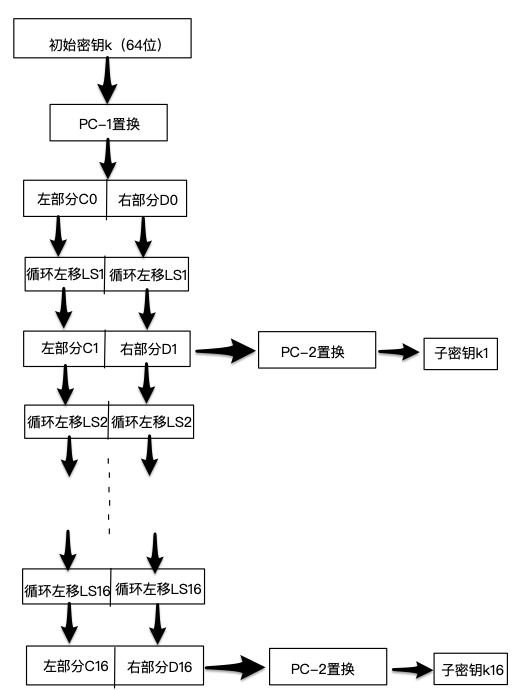
具体过程的示意图如下：  


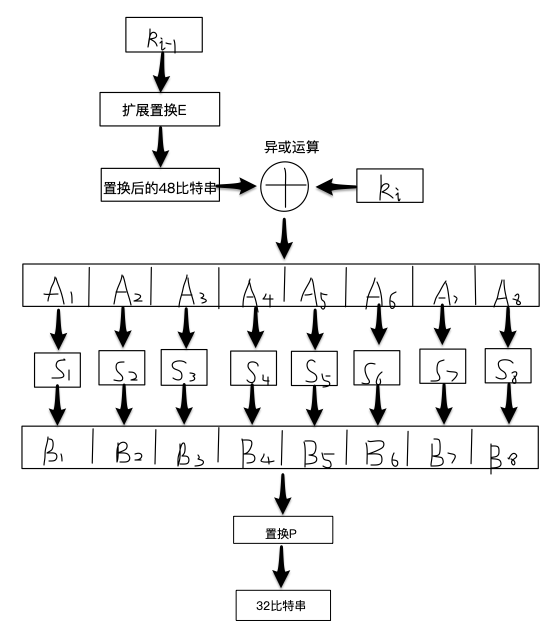
表1.2.1

| **PC-1** |
| --- |
| 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 |

表1.2.2

| **PC-2** |
| --- |
| 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 |

1.2.2 非线形函数f

f函数的参数有两个变量，一个是32比特的Ri-1，另一个是48比特的ki，输出的结果为32比特。具体执行如下图：  


过程是：

<1> Ri-1是每轮迭代运算初始值的右半部分。它的长度为32位。通过扩展置换E（表1.2.3）扩展成一个48比特的串；  
扩展置换E表里的数字都是32比特串的第几位。通过表可以看出32比特串分成8组，每组4位，然后将每组的4位前后根据表扩展，如第一组前面添加32比特串的第32位比特，后边添加32位比特串的第5位比特，其他依次类推。

<2> 48比特的串与长度一致的ki进行异或运算

<3> 将<2>步得到的48比特串分成8个6比特的串，即为A1A2A3A4A5A6A7A8

<4> 将A1，A2，A3，A4，A5，A6，A7，A8分别作为8个S盒的输入，查表（表1.2.4）得到输出B1，B2，B3，B4，B5，B6，B7，B8；  
每个S盒都是将6比特消息映射成一个4比特的消息。设Si盒的输入位6比特串x=x1x2x3x4x5x6，将x1x6转换成10进制的0～3的某个数，它对应表中的行数，将x2x3x4x5转换成0～15的10进制的某个数，它作为表的列号，利用行号和列号查询对应S盒表得到一个整数，将该整数转换成二进制就是输出结果。例如S1盒的输入是110011，则行号是11（第3行），列号是1001（第9列），查表得到整数11，再转换成二进制为1011，这就是1盒的输出结果。

<5> 将8个S盒的输出拼接一起得到B1B2B3B4B5B6B7B8，再将这32比特的串使用置换运算P（表1.2.5）得到最后的结果，也就是每轮函数f的输出。  
置换运算P：P盒置换将每一位输入位映射到输出位。任何一位都不能被映射两次，也不能被略去。映射规则是，参照置换P表，将32位的输入的第16位放在第一位，第七位放在第二位，第二十位放在第三位，以此类推。

表1.2.3  


备注：中间为32位，两侧为扩展位

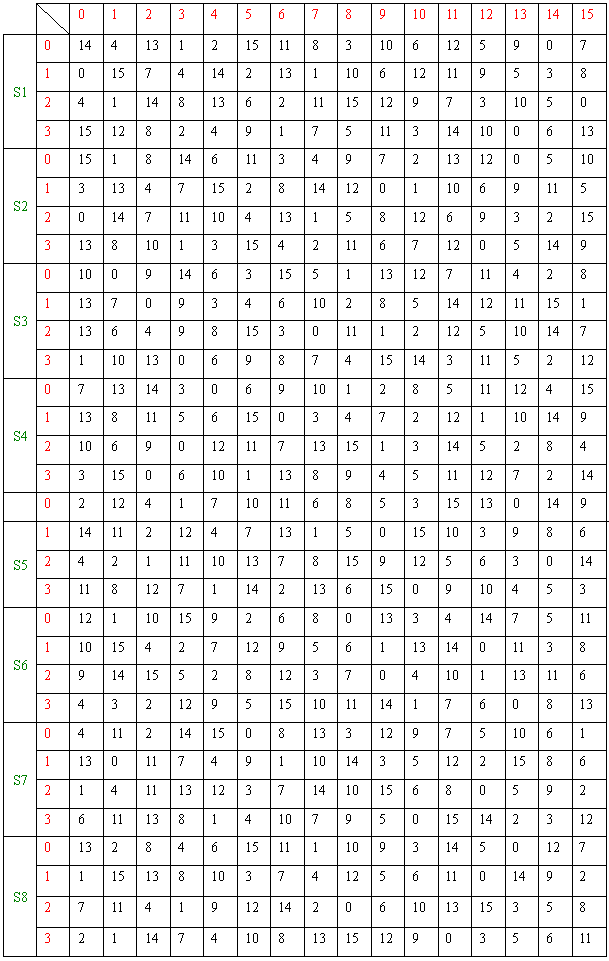
表1.2.4：  


表1.2.5：

| **置换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 |

1.3 逆初始置换IP^-1

逆置换是初始置换的逆运算。参照表1.2.6，从初始置换规则中可以看到，原始数据的第1位置换到了第40位，第2位置换到了第8位。则逆置换就是将第40位置换到第1位，第8位置换到第2位。以此类推，逆置换规则如下。

表1.2.6：

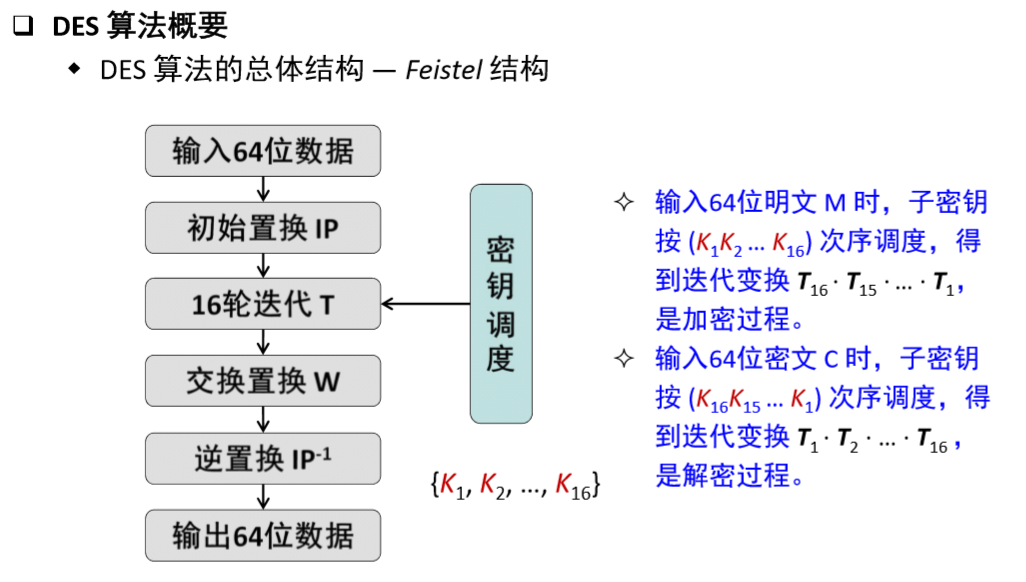
| **逆初始置换IP^-1** |
| --- |
| 40,8,48,16,56,24,64,32,39,7,47,15,55,23,63,31 |
| 38,6,46,14,54,22,62,30,37,5,45,13,53,21,61,29 |
| 36,4,44,12,52,20,60,28,35,3,43,11,51,19,59,27 |
| 34,2,42,10,50,18,58 26,33,1,41, 9,49,17,57,25 |

1.4 DES算法解密过程

加密和解密使用相同的算法。加密和解密唯一不同的是秘钥的次序是相反的。就是说如果每一轮的加密秘钥分别是K1、K2、K3...K16，那么解密秘钥就是K16、K15、K14...K1。为每一轮产生秘钥的算法也是循环的。加密是秘钥循环左移，解密是秘钥循环右移。解密秘钥每次移动的位数是：0、1、2、2、2、2、2、2、1、2、2、2、2、2、2、1。具体不做讲解。但是要注意一点，解密的结果并不一定是我们原来的加密数据，可能还含有你补得位，一定要把补位去掉才是你的原来的数据。

2 结构

2.1 总体结构



2.2 模块分解

Tables.h文件：规则表

Des.c ：计算过程

2.3 数据结构

规则表用数组表示

输入输出用十六进制数表示

输入输出的字节流用字符串存储

3 DES算法特点

1、分组加密算法：  
以64位为分组。64位明文输入，64位密文输出。

2、对称算法：  
加密和解密使用同一秘钥

3、有效密钥长度为56位  
秘钥通常表示为64位数，但每个第8位用作奇偶校验，可以忽略。

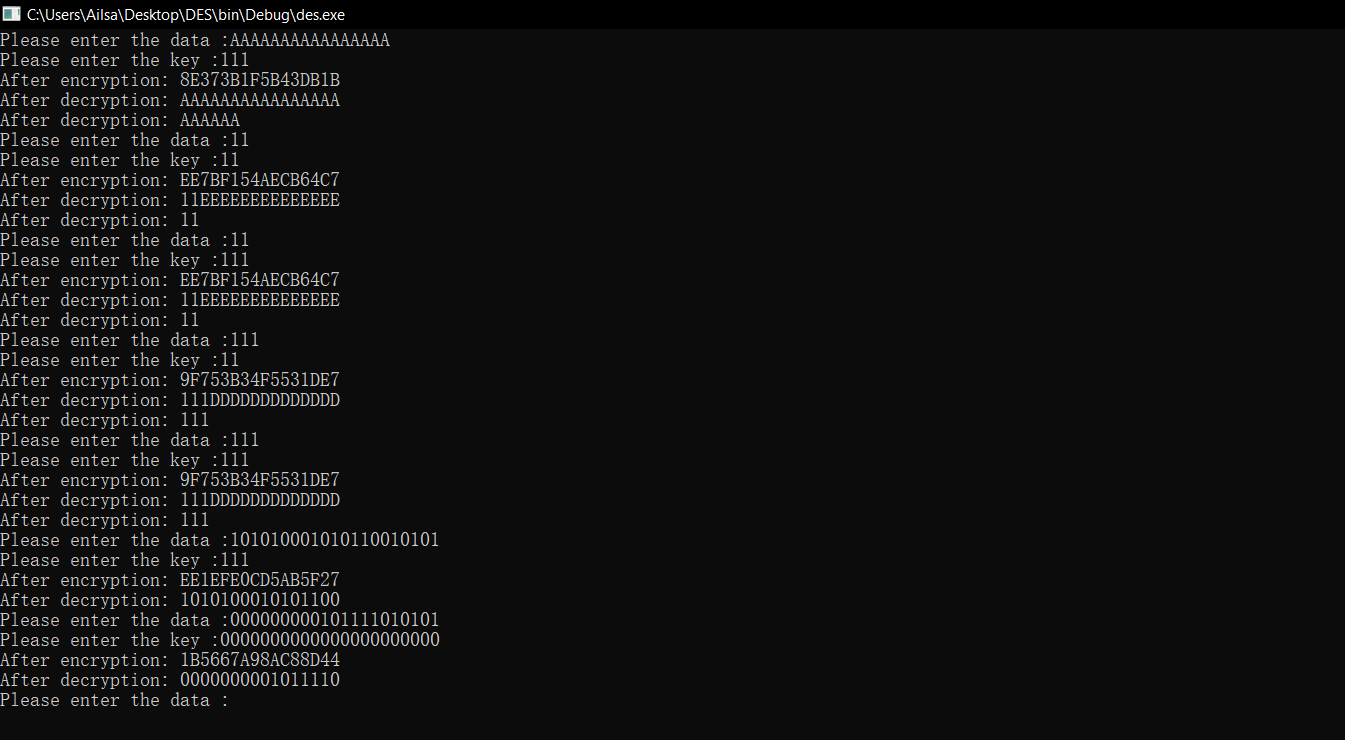
4、代替和置换  
DES算法是两种加密技术的组合：混乱和扩散。先替代后置换。

5、易于实现  
DES算法只是使用了标准的算术和逻辑运算，其作用的数最多也只有64 位，因此用70年代末期的硬件技术很容易实现

## 4 C语言实现DES算法加解密

|  |
| --- |
|  |
| #include <stdio.h>  #include <stdlib.h>  #include "string.h"  #include "tables.h"  char BitToHex(char\* bit)  {      int i,temp=0;      int ch;      for(i=0; i!=4; ++i)      {          temp<<=1;          temp+=(bit[i]&0x01);      }      if(temp>9)          ch = temp-10 + 'A';      else          ch = temp + '0';      return ch;  }  int hexCharToDec(char c)  {      int res = 0;      if(c <= 'F' && c >= 'A')          res = c-'A'+ 10;      else          res = c - '0';      return res;  }  //print HEX to screen  void printHex(char\* bitstream, int BYTES,int choose)  {      int cnt;      for (cnt=0; cnt!=BYTES; ++cnt)      {          printf("%c",BitToHex(bitstream+cnt\*4));      }      printf("\n");      if(Prune(bitstream)>0 && choose == 2)          {          printf("After decryption: ");          for (cnt=0; cnt!=BYTES-Prune(bitstream); ++cnt)          {              printf("%c",BitToHex(bitstream+cnt\*4));          }          printf("\n");          }  }  void print(char\* input){      printf("%s\n",input);  }  void HexToStr(char\* input, char\* output)  {      int temp,i;      while(\*input != '\0')      {          if(\*input<='9' && \*input >= '0')              temp=\*input-'0';          else              temp = \*input - 'A'+10;          for(i=0; i!=4; ++i)              \*(output++) = temp >> (3-i) & 1;          ++input;      }  }  /\*\*\*\*\*\*\*\*\*\*\*SUBKEY\*\*\*\*\*\*\*\*\*\*/  void Subkey\_PC1(char\* key, char\* Out)  {      //key is char[64], outL and outR are char[28]      int i;      for(i = 0; i < 28; ++i)          Out[i]=key[PC1L\_table[i]-1];      for(i = 0; i < 28; ++i)          Out[i+28]=key[PC1R\_table[i]-1];  }  void Subkey\_PC2(char\* input, char\* subkey)  {      //input: L and R, char[28]; output: subkey, char[48];      int i;      for(i=0; i < 48; ++i)          \*(subkey+i)=input[PC2\_table[i]-1];  }  void Subkey\_LeftMove(char\* In, int mvcnt)  //mvcnt stands for move times,range 1 or 2  {      char temp;      int i;      while(mvcnt--)      {          //left move once          temp=In[0];          for(i=0; i!= 27; ++i)              In[i]=In[i+1];          In[27]=temp;      }  }  void MakeSubKey(char\* key, char\* SubkeyArray)  {      //input key of char[64] and output subkeyarray[16\*48]      int i;      char PC1out[56];      Subkey\_PC1(key,PC1out); // parse key into Left and right parts      for(i=0; i!=16; ++i)      {          Subkey\_LeftMove(PC1out,MOVE\_table[i]);          Subkey\_LeftMove(PC1out+28,MOVE\_table[i]);          Subkey\_PC2(PC1out,SubkeyArray+48\*i);      }  }  /\*\*\*\*\*\*\*\*\*\*\*\*DES BODY\*\*\*\*\*\*\*\*\*\*\*\*\*/  char\* XOR(char\* In\_1, char\* In\_2, size\_t bitnum) // 'In\_1', 'In\_2': char [48], 'Out': char [48]  {      // XOR, In\_1 = Expand function's Output, In\_2 = SubKey, Out = Wait to be S\_Box      int i;      for(i = 0; i < bitnum; ++i) // 'i' is the number of bit that we have XOR.          In\_1[i]=(In\_1[i]^In\_2[i])&0x01;      return In\_1;  }  void Substitution(char\* In, char\* Out) // 'In': char [48], 'Out': char [32]  {      // The S\_Box, In = XOR's Output, Out = Wait to be P function.      int x,y,z;      int InWordLeftBound,OutWordLeftBound;      for(x = 0; x < 8; ++x)      {          InWordLeftBound = 6 \* x;    //select the input word, since we have a 48bit input and 8 boxes,          // the beginning index of each word should be 6\*x          OutWordLeftBound = x << 2;  //The same to ouputstream. But now we have only 4bit a word.          //Translate the two ends into one index.          y = (\*(In + InWordLeftBound) << 1) | (\*(In + InWordLeftBound + 5));          //translate the middle part into one index.          z = ((\*(In + InWordLeftBound + 1) << 3) | (\*(In + InWordLeftBound + 2) << 2) | (\*(In + InWordLeftBound + 3) << 1) | (\*(In + InWordLeftBound + 4)));          //Output, translate the integer into char array.          \*(Out + OutWordLeftBound) = (S\_Box[x][y][z] & 0x8) >> 3;          \*(Out + OutWordLeftBound + 1) = (S\_Box[x][y][z] & 0x4) >> 2;          \*(Out + OutWordLeftBound + 2) = (S\_Box[x][y][z] & 0x2) >> 1;          \*(Out + OutWordLeftBound + 3) = (S\_Box[x][y][z] & 0x1);      }  }  /\*\*\*\*\*\*F function begin\*\*\*\*\*\*\*/  void E(char\* In, char\* Out) // 'In': char [32], 'Out': char [48]  {      // The Expand, In = R\_i[32], Out = char[48] wait to be XOR with SubKey      int i;      for(i = 0; i < 48; ++i) // 'i' is the number of bit that we have changed.          Out[i] = In[E\_Box[i]-1];  }  void P(char\* In, char\* Out) // 'In': char [32], 'Out': char [32]  {      // F function, In = S\_Box's output, Out = Wait to be XOR with L\_i      int i;      for(i = 0; i < 32; ++i)          Out[i] = In[P\_table[i]-1];  }  void F(char\* R, char\* Subkey, char\* Out) // 'In\_1': char [32], 'In\_2':char [48], 'Out': char [32]  {      // F function,   Out = Wait to be XOR with L\_i      char Eout[48];      char SBoxout[32];      E(R, Eout);      Substitution(XOR(Eout, Subkey,48), SBoxout);      P(SBoxout, Out);  }  /\*\*\*\*\*\*\*\*Function ends\*\*\*\*\*/  void Round(char\* L, char\* R,char\* subkey) // 'InL', 'InR', 'OutL', 'OutR': char [32], 'subkey': char [48]  {      int i;      char\* temp;      char Fout[32];      for(i=0; i!=16; ++i)      {          F(R,subkey+i\*48,Fout);          temp=XOR(L,Fout,32);          L=R;          R=temp;      }  }  void IP(char\* In, char\* Out)  {      // 'In': char [16], 'Out': char [16]      // IP Transform, In = Message, Out = L\_0|R\_0      int i;      for(i = 0; i < 64; ++i) // 'i' is the number of bit that we have changed.          Out[i]=In[IP\_Box[i]-1];  #ifndef DES\_RELEASE      printf("IP layout\n");      printHex(Out, 16);  #endif  }  void FP(char\* In, char\* Out)  {      // 'In': char [64], 'Out': char [64]      // FP Transform, In = L\_16|R\_16, Out = Cipher      int i;      for(i = 0; i < 64; ++i) // 'i' is the number of bit that we have changed.      {          Out[i]=In[FP\_Box[i]-1];          //printf("%d",Out[i]);      }  }  void DES\_body(char\* block, char\* subkey, char\* out)//input: char[64], key: char[64]  {      char IPout[64],L[32],R[32];      IP(block,IPout);      memcpy(L,IPout,32);      memcpy(R,IPout+32,32);      Round(L,R,subkey);      //We have to swap L and R on the last round.      //So we can copy them on opposite side.      memcpy(IPout,R,32);      memcpy(IPout+32,L,32);      FP(IPout,out);  }  /\*input and key: char [64], output: char[64]\*/  void DES\_block\_encrypt(char\* input,char\* key,char\* output)  {      char subkey[16\*48];      MakeSubKey(key,subkey);      DES\_body(input,subkey,output);  }  void DES\_block\_decrypt\_Subkey\_Reverse(char\* subkey, char\* rev\_subkey)  {      int i;      for(i=0; i!=16; ++i)          memcpy(rev\_subkey+i\*48,subkey+(15-i)\*48,48);  }  void DES\_block\_decrypt(char\* input,char\* key,char\* output)  {      char subkey[16\*48],rev\_subkey[16\*48];      MakeSubKey(key,subkey);      DES\_block\_decrypt\_Subkey\_Reverse(subkey,rev\_subkey);      DES\_body(input,rev\_subkey,output);  }  int Prune(char \*a){      char c = BitToHex(a+15\*4);      int num = hexCharToDec(c);      int sum = 0;      for(int i = 16-num;i<16;i++)          if(BitToHex(a+i\*4) == c)              sum++;      return sum;  }  void Patch(char\* a){      int num = 16-strlen(a);      char s[16];      char c = itoa(num,s,16)[0]-32;      for(int i = strlen(a);i < 16;i++)      {          a[i] = c;      }  }  void DES\_test()  {      char datablock[64], keybyte[64];      char outputbit[64],decryptbit[64];      char databit[64],key[64];      printf( "Please enter the data :");      gets(datablock);      Patch(datablock);      printf( "Please enter the key :");      gets(keybyte);      //Process Hex to bit stream(char [64])      HexToStr(datablock,databit);      HexToStr(keybyte,key);      DES\_block\_encrypt(databit,key,outputbit);      DES\_block\_decrypt(outputbit,key,decryptbit);      //Print the result      printf("After encryption: ");      printHex(outputbit, 16,1);      printf("After decryption: ");      printHex(decryptbit, 16,2);  }  int main()  {      //test      while(1)          DES\_test();      return 0;  } |

## 5 编译运行结果



注：输入为不限长度的十六进制数，长度超过16会被自动截断，不足16会被补全为16位（也就是64位二进制）。After decryption 有两个输出结果的代表输入的密文可能有对应的两种原明文，因为信息空间补充01串之后解密的结果就会有三种可能，一种是没填充之前的原始字符串，一种是恰好末尾重复字符对应的十六进制数正好是填充的长度，比如11111111111111111，末尾是1既可以理解为补充了1个1，也可以理解为明文本来就包含了最后一个1。